Commodore 128

PERSONAL COMPUTER

Bedienungshandbuch



Bedienungshandbuch Commodore 128 Personal Computer



Gliederung des C 128-Handbuches

Inhalt

4. C128-Modus

5. C64-Modus

6. Floppy-Disk-Betrieb mit BASIC

7. CP/M-Modus – siehe CP/M-Handbuch

Allgemeine Hinweise zu diesem Handbuch

Das Bildsignal des **80-Zeichen-Modus ist nur am RGBI-Ausgang** vorhanden, nicht am Video- bzw. Fernsehausgang, d.h. bei Anschluß eines Fernsehgerätes oder eines Monitors an der Fernsehbuchse bzw. an der Videobuchse kann nur im 40-Zeichen-Modus gearbeitet werden. Umgekehrt ist der 40-Zeichen-Modus nicht am RGBI-Ausgang verfügbar.

Die mitgelieferte CP/M-Systemdiskette ist beidseitig beschrieben. Wir empfehlen Ihnen von jeder Seite eine Arbeitskopie anzufertigen und die Originaldiskette an einem sicheren Ort aufzubewahren.

1. Einführung

Machen Sie Bekanntschaft mit dem neuen C128 von Commodore, dem "großen Bruder" des weltweit bekannten C64.

Der neue C128 erlaubt Ihnen die Auswahl zwischen drei Betriebsarten, so daß er nahezu all Ihren speziellen Wünschen und Anforderungen gerecht wird.

- Der C128–Modus – Hierfür gibt es das neue BASIC 7.0. Eine um viele Funktionen und Befehle erweiterte BASIC–Version, die einen Speicher von 128 KBytes unterstützt.

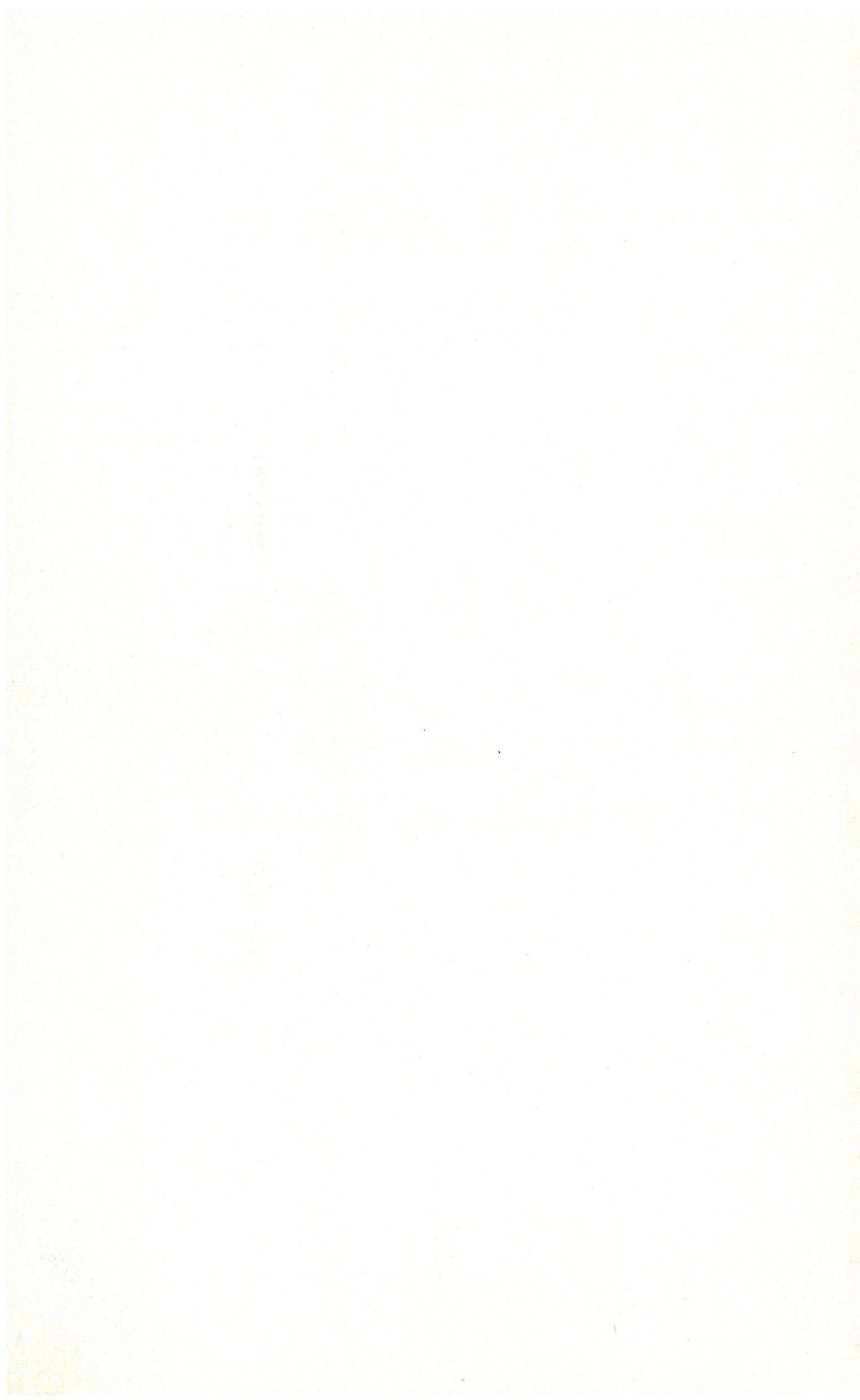
- Der C64–Modus – Ist absolut kompatibel zum C64. Es ist möglich, alle Software des C64 zu nutzen.

- Der CP/M Modus – Ermöglicht den Einsatz sämtlicher Software, die für die Betriebssysteme CP/M 3.0 und CP/M 2.2 geschrieben wurde.

Die Umschaltung zwischen den Betriebs–Modi kann programmiert erfolgen.

Aufgrund der Softwarekompatibilität zum C64 und zum Betriebssystem CP/M 3.0 ist ein fast unübersehbares Softwareangebot einsetzbar.

Damit ist der Commodore 128 einzigartig in seiner Klasse.



2. Allgemeine Eigenschaften von BASIC



2. Allgemeine Eigenschaften von BASIC

Dieses Kapitel enthält wichtige Informationen über viele grundlegende Eigenschaften des BASIC–Interpreters und sollte vor der Erstellung von BASIC–Programmen von denjenigen Anwendern beachtet werden, die mit dem Programmieren in Commodore–BASIC noch keine Erfahrung haben. Aber auch für den Fachmann enthält dieses Kapitel viel Wissenswertes.

2.1 Betriebsarten von BASIC

Der BASIC–Interpreter kennt grundsätzlich zwei Betriebsebenen: die Befehlsebene (Direktmodus) und die Programmebene.

Die **Befehlsebene** oder der **Direktmodus** ist aktiv, wenn der Rechner eingeschaltet oder wenn ein BASIC–Programm ordnungsgemäß oder durch Ausgabe einer Fehlermeldung beendet wurde. Sie ist durch die Bildschirmmeldung

```
READY.
```

und darunter wartendem Cursor (Positionsanzeiger) gekennzeichnet.

In der Befehlsebene können fast alle Befehle, Anweisungen und Funktionen des Interpreters ausgeführt werden, indem diese ohne vorangestellte Zeilennummer eingegeben werden. Drücken der RETURN–Taste schließt in jedem Fall die Eingabe ab und übergibt sie dem Interpreter zur Ausführung. Nach der Ausführung befindet sich BASIC wieder in der Befehlsebene.

Beispiel 1:

```
READY.
PRINT 5*2
10
READY.
```

Wird dem eingegebenen Befehl, der Anweisung oder Funktion jedoch eine Zahl vorangestellt, so übernimmt der Interpreter diese Zeile als Programmzeile in den Hauptspeicher und führt die darin enthaltenen Anweisungen erst nach Eingabe des RUN–Befehls oder einer auf diese Zeile bezogenen

GOTO–Anweisung aus. Auf diese Weise können Programme eingegeben werden.

Beispiel 2:

```
READY.
10 PRINT "ERGEBNIS IST";
20 PRINT 5*2
RUN
ERGEBNIS IST 10
READY.
```

In der **Programmebene** befindet sich der Interpreter, sobald der Anwender das eingegebene Programm gestartet hat. Der Cursor (Positionsanzeiger) verschwindet dann (Ausnahme ist hier die INPUT–Anweisung für die Dateneingabe über Tastatur) und erscheint erst wieder, wenn der Interpreter nach der Programmausführung wieder auf die Befehlsebene zurückkehrt (s. oben Beispiel 2).

2.2 BASIC–Zeilenformat und –Zeichenvorrat

Ein BASIC–Programm besteht grundsätzlich aus Programmzeilen, die das folgende Format haben müssen:

nnnnn Befehl [:***Befehl. . .***][:**REM** ***Kommentar***]

nnnnn Eine Zeilennummer als ganze Zahl zwischen 0 und 63999, die die Reihenfolge der Programmzeilen im Hauptspeicher festlegt und auch als Bezugsnummer für Programmverzweigungen dient.

Befehl Ein beliebiger BASIC–Befehl oder eine beliebige BASIC–Anweisung, –Funktion oder –Variable (s. Kapitel 4.6 und 5.2). Mehrere solcher Befehle in einer Zeile müssen durch Doppelpunkte (:) voneinander getrennt werden.

Kommentar Freier Text zur Kommentierung des Programms, der nach einer REM–Anweisung folgt.

- Anmerkung: Die Gesamtlänge einer BASIC–Programmzeile darf 160 Zeichen nicht überschreiten.
(Im C64–Modus max. 80 Zeichen)

Neben den Groß– oder Kleinbuchstaben des lateinischen Alphabets sowie den Ziffern 0 bis 9 kennt BASIC eine Reihe von Sonderzeichen, die eine besondere Bedeutung haben:

Zeichen	Bedeutung
+	Plussymbol oder Verkettungssymbol für Zeichenketten
–	Minussymbol oder Bindestrich
*	Multiplikationssymbol oder Stern
/	Divisionssymbol oder Schrägstrich
↑	Potenzierungssymbol
=	Gleichheitszeichen oder Zuweisungssymbol
<	Kleiner–als–Symbol oder linke spitze Klammer
>	Größer–als–Symbol oder rechte spitze Klammer
(	linke Klammer
)	rechte Klammer
[	linke eckige Klammer
]	rechte eckige Klammer
%	Typsymbol für ganze Zahl oder Prozentzeichen
#	Nummernzeichen
$	Typsymbol für Zeichenketten oder Dollarzeichen
.	Dezimalpunkt oder Punkt
,	Tabulierungszeichen oder Komma
:	Trennzeichen für Befehle oder Doppelpunkt
;	Zeilenvorschubunterdrückung oder Semikolon
"	Zeichenketten–Anfang bzw. –Ende oder Anführungszeichen
?	Abkürzung für PRINT–Anweisung oder Fragezeichen

Alle diese Zeichen können auch als normale druckbare Zeichen Bestandteil von Zeichenketten sein. Weitere druckbare Zeichen sind in der vollständigen ASCII–Zeichencodetabelle im Anhang A zusammengestellt.

2.3 Reservierte Wörter

Die für den BASIC–Interpreter reservierten Wörter sind die Bezeichnungen für die verschiedenen Befehle, Anweisungen, Funktionen, Variablen und logischen Operatoren. Sie dürfen nicht als Variablennamen verwendet

werden. Im folgenden sind alle reservierten Wörter alphabetisch zusammengestellt.

ABS
AND
APPEND
ASC
ATN
AUTO

BACKUP
BANK
BEGIN
BEND
BLOAD
BOOT
BOX

BSAVE
BUMP

CATALOG
CHAR
CHR$
CIRCLE
CLOSE
CLR
CMD
COLLECT
COLLISIOn
COLOR
CONCAT
CONT
COPY

DATA
DCLEAR
DCLOSE
DEC
DEF
DELETE
DIM
DIRECTORY
DLOAD
DO
DOPEN
DRAW
DS
DSAVE
DS$
DVERIFY

EL
ELSE
END
ENVELOPE
ER
ERR$
EXIT
EXP

FAST
FETCH
FILTER
FNxx
FOR
FRE

GET
GETKEY
GO64
GOSUB
GOTO
GRAPHIC
GSHAPE

HEADER
HELP
HEX$

IF
INSTR
INPUT
INPUT#
INT

JOY

KEY

LEFT$
LEN
LET
LIST
LOAD
LOCATE
LOG
LOOP

MID$
MONITOR
MOVSPR

NEW
NEXT
NOT

ON
OPEN
OR

PAINT
PEEK
PEN

PLAY
POINTER
POKE
POS
POT
PRINT
PRINT#
PUDEF
RCLR
RDOT
READ
RECORD
REM
RENAME
RENUMBER
RESTORE
RESUME
RETURN
RGR
RIGHT$
RND
RREG
RSPCOLOR
RSPPOS
RSPRITE
RUN
RWINDOW

SAVE
SCALE
SCNCLR
SCRATCH
SGN
SIN
SLEEP
SLOW
SOUND
SPC
SPCOLOR
SPRDEF
SPRITE
SPRSAV
SQR
SSHAPE
ST
STASH
STEP
STOP

Reservierte Wörter (Fortsetzung)

SWAP	TI	UNTIL	WAIT
SYS	TI$	USING	WHILE
	TO	USR	WIDTH
TAB	TRAP		WINDOW
TAN	TRON	VAL	
TEMPO	TROFF	VERIFY	XOR
THEN		VOL	

2.4 BASIC–Konstanten

Konstanten sind aktuelle Werte, die der BASIC–Interpreter während der Programmausführung verwendet. Er unterscheidet dabei zwischen Zeichenketten–Konstanten und numerischen Konstanten.

Eine **Zeichenkettenkonstante** ist eine Folge von bis zu 255 beliebigen (alphanumerischen) Zeichen, die in Anführungszeichen (") eingeschlossen ist. Z.B.:

"HAUS"
"Hund und Katze"
"DM 12.80"

Numerische Konstanten sind positive oder negative Zahlen, die außer dem Vorzeichen nur einen Dezimalpunkt (**kein** Komma) enthalten dürfen. Der Interpreter unterscheidet drei verschiedene Arten von numerischen Konstanten:

Ganzzahlige Konstanten sind ganze Zahlen im Bereich zwischen –32768 und +32767 ohne Dezimalpunkt. Z.B.:

125

Festkommakonstanten sind alle positiven oder negativen reellen Zahlen mit einem Dezimalpunkt. Z.B.:

15.12

Gleitkommakonstanten sind alle positiven oder negativen Zahlen in Exponentialdarstellung (wissenschaftliche Darstellung). Sie bestehen aus einer ganzen oder einer Festkommazahl (Mantisse) mit oder ohne Vorzeichen,

gefolgt vom Buchstaben E und einer ganzen Zahl im Bereich zwischen –39 und +38 mit oder ohne Vorzeichen, die den Exponenten zur Basis 10 darstellt. Z.B.:

15E–3	gleichbedeutend mit 0.015
146.285E–6	gleichbedeutend mit 0.000146285

Der Interpreter akzeptiert Gleitkommakonstanten im Bereich von 1.7E+38 bis 2.9E–39 und –2.9E–39 bis –1.7E+38.

2.5 BASIC–Variablen

Variablen sind Namen, die Werte repräsentieren, welche in einem BASIC-Programm verwendet werden. Der Interpreter unterscheidet dabei zwischen

Zeichenketten–Variablen
numerischen Variablen

Zeichenkettenvariablen dürfen aus bis zu 255 Zeichen bestehen. Die Anzahl dieser Zeichen bestimmt die Länge des Variablenwertes.

Numerische Variablen haben immer einen Wert, der aus einer Zahl besteht.

Der Wert einer Variablen kann dieser explizit vom Programmierer oder, als Ergebnis von Programmoperationen, vom Programm zugewiesen werden. Werden Variablen benutzt, ehe ihnen ein Wert zugewiesen wurde, ist ihr Wert im Fall von numerischen Variablen 0 und im Fall von Zeichenkettenvariablen eine leere Zeichenkette der Länge 0 (Leer–String).

2.5.1 Variablennamen und –typen

Der BASIC–Interpreter akzeptiert beliebig lange Variablennamen, interpretiert jedoch nur die ersten 2 Zeichen sowie ggf. das letzte, das den Variablentyp definiert (s. u.) und eines der Sonderzeichen $ oder % sein muß, wenn ein anderer Typ als Gleitkomma definiert werden soll.

Ein Variablenname muß als erstes Zeichen einen Buchstaben haben und darf aus Buchstaben und Ziffern in jeder beliebigen Kombination bestehen. Er darf kein reserviertes BASIC–Wort (s. Kapitel 2.4) sein.

Beispiele:

`SIN=101`	verboten (SIN–Funktion)
`FN$="NAME"`	verboten (Benutzerfunktion DEFFN)

Der Name einer Variablen dient nicht nur ihrer Benennung, sondern auch der Definition ihres Typs.

Zeichenkettenvariablen werden durch ein $ als letztem Zeichen im Namen definiert. Z.B.:

C$ = "COMMODORE"

Numerische Ganzzahlvariablen (Typ integer) werden durch ein % als letztem Zeichen im Namen definiert:

TAG% = 30

Numerische Gleitkommavariablen (Typ real) haben kein Sonderzeichen am Namensende:

ZINS = 15.84

Je nach Art der numerischen Variablen ist der Platzbedarf bei

Ganzzahlvariablen	2 Bytes
Gleitkommavariablen	5 Bytes (10 Stellen)

2.5.2 Feldvariablen

Ein Feld ist eine Gruppe oder Tabelle von numerischen oder Zeichenkettenwerten, die durch einen Variablennamen repräsentiert wird. Jeder dieser Werte wird als Feldelement bezeichnet und mit einem Index zum Variablennamen beschrieben. Dieser Index ist eine ganze Zahl oder ein ganzzahliger Ausdruck. Einem Feldvariablennamen sind genausoviele Indizes zugeordnet, wie das Feld Dimensionen hat. Die Dimensionierung eines Feldes, mit der gleichzeitig auch die Benennung verbunden ist, erfolgt mit der Anweisung DIM (s. dort in Kapitel 5.2). Beispiele:

`DIMX(25)`	Ein eindimensionales Gleitkommafeld X mit 26 Elementen (0 bis 25) wird dimensioniert.
`DIMX$(2,6)`	Ein zweidimensionales Zeichenkettenfeld mit 21 Elementen (3*7) wird dimensioniert.

Mit der Dimensionierung werden alle Feldelemente des dimensionierten Feldes auf Null gesetzt.

Der Index bestimmt die Position eines Elementes im Feld.

Die maximale Anzahl von Elementen in jeder Dimension ist 32767 und es sind maximal 255 Dimensionen möglich. Auf jeden Fall begrenzt der verfügbare Speicher die Größe eines Feldes.

Bei zweidimensionalen Feldern gibt der erste Index grundsätzlich die Zeile und der zweite Index die Spalte der durch ein solches Feld gebildeten Tabelle an.

Eindimensionale Felder mit bis zu 11 Elementen brauchen nicht dimensioniert zu werden. Wird in einem Programm z.B. die Anweisung

```
A(6)=12.3
```

verwendet, ohne daß vorher A dimensioniert wurde, so führt der Interpreter intern die Anweisung

```
DIMA(10)
```

aus und dimensioniert damit das Feld selbst.

Für die Namensvergabe und den Typ gelten dieselben Regeln wie für einfache Variablen (s. Kapitel 2.5.1).

2.6 Genauigkeit numerischer Werte

Numerische Werte können, wie bereits erwähnt, intern als ganze Zahlen oder als reelle Zahlen gespeichert werden.
Ganzzahlige Werte werden in zwei Bytes des Hauptspeichers abgelegt.
Reelle Werte werden als Gleitkommazahlen mit einer Genauigkeit bis zu 10 Stellen gespeichert.

Konstanten, die keine ganzen Zahlen sind, können folgendermaßen geschrieben werden:

- bis zu 10 Zeichen z.B. 100.59
- Exponentialdarstellung mit E z.B. 3.57E–05

2.7 Numerische Ausdrücke und Operatoren

Ein numerischer Ausdruck kann eine numerische Konstante oder Variable oder eine Kombination aus numerischen Konstanten, Variablen und Operatoren zur Berechnung eines einzelnen Wertes sein. Die Operatoren führen dabei mit den Werten mathematische oder logische Operationen durch. Der BASIC–Interpreter kennt vier verschiedene Kategorien von Operatoren:

- arithmetische Operatoren
- Vergleichsoperatoren
- logische Operatoren
- Funktionsoperatoren

2.7.1 Arithmetische Operatoren

Es gibt sechs arithmetische Operatoren. Es sind dies in der Reihenfolge ihrer Berücksichtigung (mathematische Hierarchie) in einem numerischen Ausdruck:

Operator	Operation	Beispiel
↑	Potenzierung	A↑5
–	Negation	–A
*,/	Multiplikation, Division	A*5, A/5
+,–	Addition, Subtraktion	A+5, A–5

2.7.2 Vergleichsoperatoren

Vergleichsoperatoren dienen dem Vergleich zweier numerischer oder Zeichenkettenwerte. Das Ergebnis ist entweder logisch "wahr" mit dem Wert –1 oder logisch "falsch" mit dem Wert 0 (Null) und wird meistens in Verbindung mit der IF–Anweisung (s. dort in Kapitel 5.2) zur Steuerung des Programmablaufes verwendet.

Im folgenden sind die sechs verschiedenen Vergleichsoperatoren, die der BASIC–Interpreter kennt, tabellarisch zusammengestellt.

Operator	Vergleich auf	Beispiel
=	Gleichheit	A=B
<>,><	Ungleichheit	A<>B, A><B
<	kleiner als	A<B
>	größer als	A>B
<=,=<	kleiner als oder gleich	A<=B, A=<B
>=,=>	größer als oder gleich	A>=B, A=>B

Wenn in einem Ausdruck sowohl arithmetische als auch Vergleichsoperatoren vorkommen, so werden zuerst die arithmetischen Operatoren abgearbeitet.

Beispiele:

```
A+B<(X-Y)/Z
```

Dieser Ausdruck ist "wahr" (liefert den Wert –1), wenn der Wert von A+B kleiner ist als der Wert X–Y dividiert durch Z.

```
100 IF SIN(X)<0 THEN 1000
```

Wenn der SIN(X) negativ wird, liefert der Vergleich ein "wahres" Ergebnis (–1) und das Programm verzweigt nach Zeile 1000.

Bei Zeichenkettenvergleichen wird von beiden Ketten zeichenweise der CBM–ASCII–Code verglichen. Es werden alle Zeichen, also auch führende und nachfolgende Leerstellen verglichen. Die Wirkung ist am besten an folgenden Beispielen zu erkennen, die alle logisch "wahr" sind, also den Wert –1 liefern:

```
"AA"<"AB"
"Aa"<"AA"
"HAUS"="HAUS"
"HAUS">"HANS"
"A!"<"A$"
"U ">"U"
X$="12":X$<"13"
```

2.7.3 Logische Operatoren

Logische Operatoren dienen zum Testen von Mehrfachvergleichen, zur Bit–Manipulation oder zum Durchführen Boolescher Operationen mit numerischen Werten.

Ein logischer Operator verknüpft zwei Operanden als Kombination aus "wahr"– und "falsch"–Werten bitweise und liefert als Ergebnis einen Wert, der entweder als "wahr" (von Null verschieden) oder als "falsch" (Null) interpretiert wird.

Der Interpreter führt Operationen mit logischen Operatoren in einem gemischten Ausdruck **nach** den arithmetischen und den Vergleichsoperationen durch.

Der Interpreter kennt drei verschiedene logische Operatoren, die im folgenden in der Reihenfolge, wie sie bearbeitet werden, mit ihren Wahrheitstabellen dargestellt werden:

Operator	Operand 1	Operand 2	Ergebnis
NOT (log. Komplement)	wahr	–	falsch
	falsch	–	wahr
AND (Konjunktion)	wahr	wahr	wahr
	wahr	falsch	falsch
	falsch	wahr	falsch
	falsch	falsch	falsch
OR (Disjunktion)	wahr	wahr	wahr
	wahr	falsch	wahr
	falsch	wahr	wahr
	falsch	falsch	falsch

Genau wie die Vergleichsoperatoren über ihr Ergebnis zur Steuerung des Programmablaufes beitragen können, kann dies auch durch die Verknüpfung von zwei oder mehreren Vergleichen durch logische Operatoren geschehen, die wiederum "wahr"– und "falsch"–Werte liefern (s. a. IF–Anweisung in Kapitel 5.2).

Die folgenden Beispiel–Programmzeilen sollen dies verdeutlichen:

```
100 IF D<200 AND F<4 THEN 80
110 IF I>10 OR K<0 THEN 50
120 IF NOT P THEN 100
```

Die logischen Operatoren arbeiten intern folgendermaßen:

Zunächst werden die beiden Operanden in ganze, vorzeichenbehaftete Zweierkomplement–16–Bit–Zahlen im Bereich zwischen –32768 und +32767 umgewandelt. Sind die Operanden größer oder kleiner, so wird die Fehlermeldung

`ILLEGAL QUANTITY ERROR` (unerlaubter Betrag)

angezeigt.

Sind die beiden Operanden 0 und/oder –1, so liefert eine logische Operation ebenfalls 0 oder –1. Die Operation wird auf jeden Fall bitweise durchgeführt, d. h. jedes Ergebnis–Bit wird durch die beiden entsprechenden Bits in den Operanden bestimmt. Dabei bedeutet ein 1–Bit "wahr" und ein 0–Bit "falsch". Dadurch ist es möglich, mit Hilfe der logischen Operatoren das Bitmuster von Speicherzellen zu testen. Z.B. kann ein Byte an einer Ein–/Ausgabeschnittstelle maskiert werden, um den Zustand eines bestimmten Bits zu testen (s.a. WAIT–Anweisung in Kapitel 5.2). Auch kann mit dem OR–Operator in einem bestimmten Byte ein ganz bestimmtes Bitmuster erzeugt werden. Die folgenden Beispiele sollen die Arbeitsweise der logischen Operatoren erläutern:

63 AND 16 = 16		0000000000111111	63
	AND	0000000000010000	16
		0000000000010000 =	16
15 AND 14 = 14		0000000000001111	15
	AND	0000000000001110	14
		0000000000001110 =	14
–1 AND 8 = 8		1111111111111111	–1
	AND	0000000000001000	8
		0000000000001000 =	8

4 OR 2 = 6		0000000000000100	4
	OR	0000000000000010	2
		0000000000000110 =	6
10 OR 10 = 10		0000000000001010	10
	OR	0000000000001010	10
		0000000000001010 =	10
–1 OR –2 = –1		1111111111111111	–1
	OR	1111111111111110	–2
		1111111111111111 =	–1
NOT 1 = –2		0000000000000001	1
	NOT	1111111111111110 =	–2

Der Operator NOT bildet das Einerkomplement des Operanden.

2.7.4 Funktionsoperatoren

Funktionsoperatoren (BASIC–Funktionen) werden dazu verwandt, an einem Operanden eine festgelegte Operation auszuführen. Solche im BASIC–Interpreter enthaltenen Funktionen sind z.B. SQR (Quadratwurzel ziehen) oder SIN (trigonometrischer Sinus). Es ist auch möglich, mit Hilfe der DEF FN–Anweisung eigene Funktionen zu definieren. Einzelheiten zu den Funktionen sind in den Kapiteln 4.6 und 5.2 ausführlich beschrieben.

2.7.5 Mathematische Hierarchie

Bei der Bearbeitung numerischer Ausdrücke und Operatoren hält der BASIC–Interpreter folgende mathematische Hierarchie ein (1 höchste, 8 niedrigste Priorität):

1. Funktionsaufrufe
2. Potenzierung (^)
3. Negation (Wandlung von Plus nach Minus u. umgekehrt)
4. Multiplikation und Division (*, /)

5. Addition und Subtraktion (+, –)
6. logisches Komplement (NOT)
7. Konjunktion (AND)
8. Disjunktion (OR)

Aufeinanderfolgende Operatoren derselben Hierarchiestufe werden von links nach rechts abgearbeitet. Die Hierarchie kann durch Verwendung von Klammern aufgehoben werden. Geklammerte Ausdrücke werden grundsätzlich zuerst bearbeitet.

Bei geschachtelten Klammerausdrücken wird grundsätzlich der innerste Klammerausdruck zuerst bearbeitet.

Folgende Beispiele verdeutlichen dies:

4+1*2	ergibt 6
(4+1)*2	ergibt 10
100*4/2–1	ergibt 199
100*(4/2–1)	ergibt 100
100*(4/(2–1))	ergibt 400

2.8 Zeichenkettenausdrücke und Operationen mit Zeichenketten

Ein Zeichenkettenausdruck kann eine Zeichenkettenkonstante oder –Variable oder eine Kombination aus Zeichenkettenkonstanten, –variablen und –operatoren zur Erzeugung einer neuen Zeichenkette sein. Die Operatoren kombinieren dabei Zeichenketten zu neuen Zeichenketten. Der BASIC–Interpreter kennt zwei Typen von Zeichenkettenoperatoren:

- Verkettungsoperator
- Funktionsoperator oder Zeichenkettenfunktion

Da der Zeichenkettenvergleich prinzipiell ein numerischer Vergleich ist, ist er im Kapitel 2.7.2 bei den Vergleichsoperatoren beschrieben.

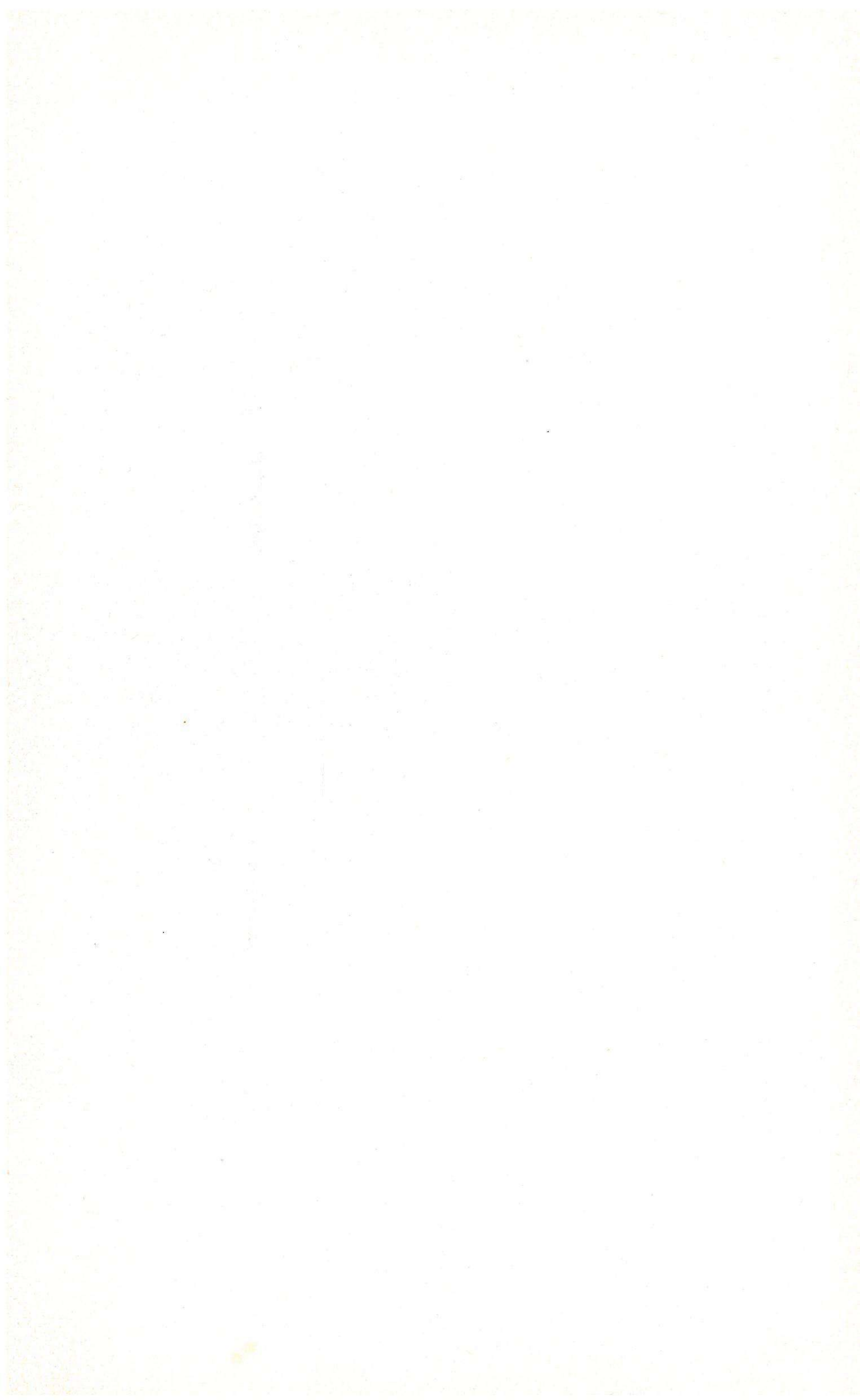
2.8.1 Verkettungsoperator

Zeichenketten können mit dem Plus–Zeichen (+) aneinandergefügt werden, um neue Zeichenketten zu bilden. Z.B.:

```
10 A$="REGEN":B$="SCHIRM"
20 PRINT A$+B$+"E"
RUN
REGENSCHIRME
READY.
```

2.8.2 Zeichenkettenfunktionen

Wie bei den numerischen Funktionen (s. Kapitel 2.7.4) führt die Zeichenkettenfunktion an einem oder mehreren Operanden eine festgelegte Operation aus, die als Ergebnis eine Zeichenkette liefert. Solche im BASIC–Interpreter enthaltenen Funktionen sind z.B. CHR$ (Bildung einer Ein–Zeichenkette aus einem ASCII–Code) oder STR$ (Bildung eines Zeichenkettenäquivalents zu einem numerischen Ausdruck). Es ist jedoch nicht möglich, eigene Zeichenkettenfunktionen mit Hilfe der DEF FN–Anweisung zu definieren. Einzelheiten zu den Zeichenkettenfunktionen sind in den Kapiteln 4.6 und 5.2 ausführlich beschrieben.



3. Eingeben und Verwalten von BASIC-Programmen



3. Eingeben und Verwalten von BASIC–Programmen

3.1 Tastatur

Abbildung 3.1 auf der nächsten Seite zeigt die Tastatur des C128. Im C128–Modus können alle Tasten verwendet werden, im C64–Modus nur die schraffierten Tasten. In diesem Abschnitt werden alle die Tasten beschrieben, die in beiden Modi verwendet werden können, also die schraffierten Tasten.

Tastaturmodi: Die Tastatur des C128 erlaubt zwei Betriebsmodi, nämlich

- Großbuchstaben/Grafik–Symbole
- Kleinbuchstaben/Großbuchstaben.

Nach dem Einschalten des Rechners ist der Großbuchstaben/Grafik–Modus aktiv. Um zwischen den beiden Modi hin–und herzuschalten, werden die Umschalttaste (SHIFT) und die C=–Taste (Commodore–Taste) zusammen gedrückt.

Schreibmaschinentastatur: Den Hauptteil der Tastatur nimmt das Schreibmaschinen–Tastenfeld ein, das auch so funktioniert.

Neben den Buchstaben– und Zeichentasten gibt es eine Reihe von Bedienungstasten, deren Funktion teilweise anders ist als bei der Schreibmaschine.

Umschalttasten: Die Umschalttasten (SHIFT–Tasten) dienen je nach aktivem Tastaturmodus zur Eingabe von Großbuchstaben oder den Sonderzeichen, die auf den doppelt belegten Tasten oben abgebildet sind (z.B. < >) bzw. von Grafiksymbolen, die vorne auf den Buchstabentasten abgebildet sind. Entweder die linke oder die rechte Umschalttaste muß zusammen mit der entsprechenden Zeichentaste gedrückt werden. Die Umschaltung auf Großbuchstaben kann auch mit der Feststeller–Taste

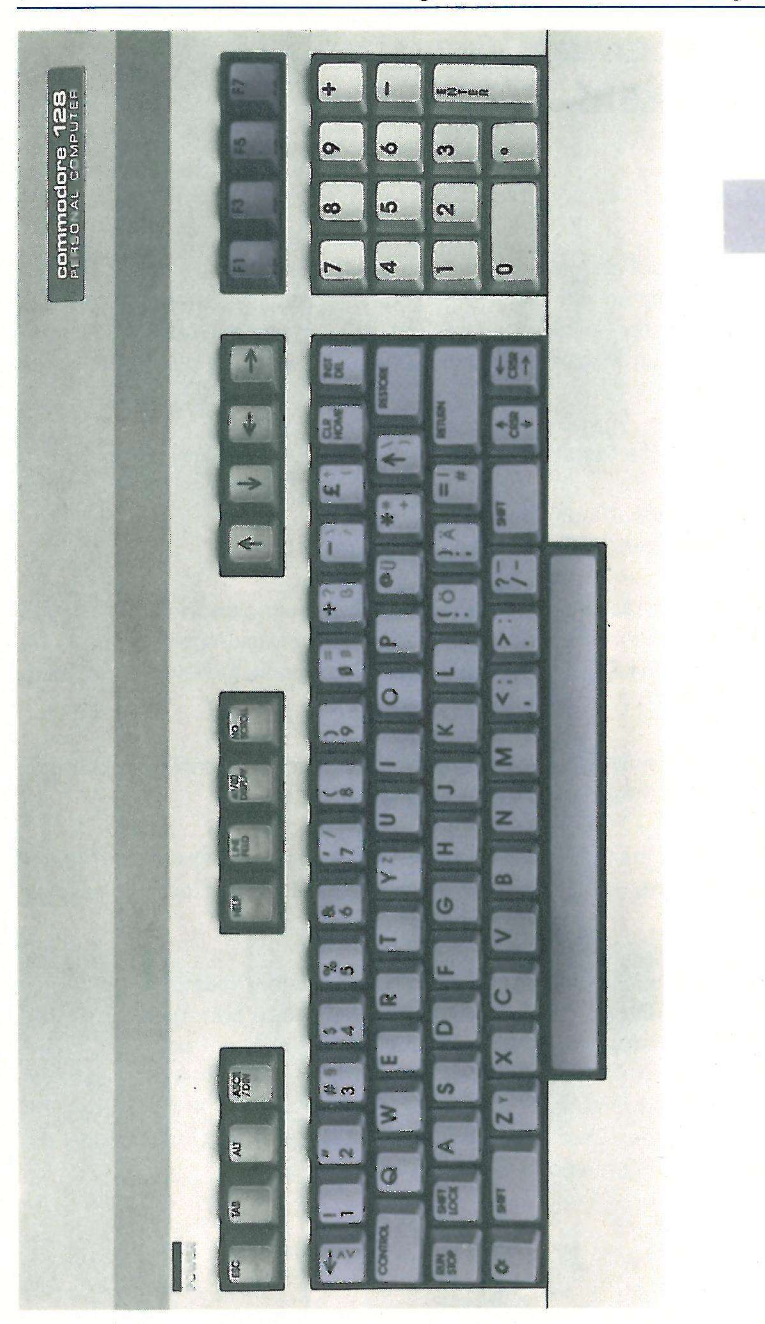


Abb. 3.1: Tastatur des C128

(SHIFT LOCK) verriegelt werden.

RETURN-Taste: Die Return-Taste entspricht der Wagenrücklauftaste bei der elektrischen Schreibmaschine. Sie schließt die Eingabe einer Zeile ab.

RESTORE-Taste: Die RESTORE-Taste dient zusammen mit der RUN/STOP-Taste dazu, den Rechner auf seine Standardparameter (dunkelgrauer Bildschirm mit hellgrünem Rand (C128-Modus) bzw. blauer Bildschirm (C64-Modus) sowie Großbuchstaben/Grafikzeichen-Tastaturmodus) zurückzusetzen.

CONTROL-Taste: Die CONTROL-Taste oberhalb der RUN/STOP-Taste dient zum Eingeben spezieller Zeichen sowie zum Auslösen von besonderen akustischen Signalen, Bildschirm- oder Systemfunktionen. Auch hier wird die CONTROL-Taste niedergehalten und eine weitere Taste gedrückt. Eine häufig verwendete Funktion der CONTROL-Taste ist die Einstellung einer Farbe. Zu diesem Zweck werden die CONTROL-Taste und eine der Zifferntasten in der oberen Reihe des Schreibmaschinen-Tastenfeldes gedrückt. Auf der Frontseite dieser Tasten sind die jeweiligen Farben (Zeichenfarbe im Textmodus) vermerkt.

RUN/STOP-Taste: Drücken dieser Taste ohne SHIFT-Taste unterbricht die augenblicklich vom Rechner ausgeführte BASIC-Anweisung und setzt ihn in den Direktmodus. Zusammen mit der SHIFT-Taste wird das erste auf Kassette (C64-Modus) oder Diskette (C128-Modus) befindliche Programm geladen und gestartet.

C= (Commodore)-Taste: Diese Sondertaste hat zwei Funktionen:

1. Zusammen mit der SHIFT-Taste kann mit dieser Taste zwischen den beiden bereits erwähnten Tastatur-Modi hin- und hergeschaltet werden.
2. Zusammen mit den Zifferntasten in der oberen Reihe der Schreibmaschinentastatur kann eine Farbe aus einem zweiten Satz von acht Farben ausgewählt werden.
3. Im C128-Modus können bei eingeschalteter DIN-Tastatur Grafikzeichen eingegeben werden (s. Anhang D).

CLR/HOME-Taste: Drücken der HOME-Taste setzt den Cursor in die linke obere Ecke des Bildschirms. Wird gleichzeitig die SHIFT-Taste niedergehalten, so wird zusätzlich der Bildschirm gelöscht.

INST/DEL-Taste: Ohne gleichzeitiges Drücken der SHIFT-Taste wird das Zeichen links neben dem Cursor gelöscht. Alle Zeichen rechts vom gelösch-

ten Zeichen werden, falls vorhanden, um eine Stelle nach links versetzt. Bei gleichzeitigem Drücken der SHIFT–Taste wird an der Cursor–Position eine Leerstelle eingefügt (Einfügemodus). Alle Zeichen rechts davon werden, falls vorhanden, um eine Stelle nach rechts versetzt.
Bei diesem Einfügemodus wird außerdem die direkte Wirkung der Cursor-Steuertasten aufgehoben und statt dessen ihr Steuercode in Form eines inversen Zeichens auf dem Bildschirm angezeigt.

CRSR–Taste: Mit SHIFT–Taste wird der Cursor in dieselbe Spalte der darüberliegenden Zeile gesetzt. In der obersten Zeile des Bildschirms hat diese Taste keine Funktion. Ohne SHIFT–Taste wird der Cursor in dieselbe Spalte der darunterliegenden Zeile gesetzt. In der untersten Zeile des Bildschirms bewirkt diese Taste ein Aufrollen des Bildschirms um eine Zeile.

CRSR–Taste: Ohne SHIFT–Taste wird der Cursor um eine Stelle nach rechts in derselben Bildschirmzeile gesetzt. Steht er vorher am Zeilenende, so wird er auf den Anfang der folgenden Zeile gesetzt. In der rechten unteren Ecke des Bildschirms führt diese Taste zum Auf rollen des ganzen Bildschirms um eine Zeile. Mit der SHIFT–Taste wird der Cursor um eine Stelle nach links in derselben Bildschirmzeile versetzt. Steht er vorher am Zeilenanfang, so wird er ans Ende der vorhergehenden Zeile gesetzt. In der linken oberen Ecke des Bildschirms hat diese Taste keine Funktion.

Funktionstasten: Den acht Funktionstasten f1 bis f8, die in einem Block oberhalb der separaten Zehnertastatur angeordnet sind, sind im C128–Modus bestimmte Zeichenfolgen (häufig benötigte BASIC–Befehle und –Anweisungen) zugeordnet, die beim Drücken der jeweiligen Taste auf dem Bildschirm angezeigt werden. Die Belegung dieser Tasten kann vom Anwender jedoch jederzeit für eigene Zwecke geändert werden (s. Kapitel 4.1, KEY–Anweisung in Kapitel 4.6 sowie Anhang G).

3.2 Eingeben von Programmzeilen

Der BASIC–Interpreter faßt jede Textzeile, die mit einer Zahl beginnt, als Programmzeile auf, wobei die Zahl die Zeilennummer darstellt.

Eine BASIC–Programmzeile beginnt also immer mit einer Zahl, wird durch die Return–Taste beendet und darf maximal 160 Zeichen einschließlich des der RETURN–Taste zugeordneten Zeichencodes enthalten. Werden mehr Zeichen eingegeben, so werden diese beim Drücken der RETURN–Taste abgeschnitten, obwohl sie durch die Eingabe noch auf dem Bildschirm stehen.

Alle BASIC–Schlüsselwörter und –Variablennamen müssen bei aktiviertem Kleinbuchstaben/Großbuchstaben–Tastaturmodus grundsätzlich in Kleinbuchstaben eingegeben werden.

Wird bei der Eingabe einer Programmzeile das Fragezeichen (?) als Abkürzung für die PRINT–Anweisung (s. dort in Kapitel 3) verwendet, so wandelt der Interpreter dies beim LIST–Befehl in das Schlüsselwort PRINT um, wodurch die Zeile ggf. länger als 80 Zeichen wird. In einem solchen Fall darf die Zeile länger als 80 Zeichen sein.

Wird eine Programmzeile mit einer Zeilennummer eingegeben, die kleiner ist als die bis dahin eingegebene größte Zeilennummer, so fügt der Interpreter diese Zeile an der richtigen Stelle ins bereits eingegebene Programm ein. Programmzeilen brauchen also nicht in der Reihenfolge aufsteigender Zeilennummern eingegeben zu werden.

Wird eine Programmzeile mit einer Zeilennummer eingegeben, unter der bereits eine andere Programmzeile existiert, so wird die alte von der neuen Programmzeile überschrieben.

Hinzufügen einer Programmzeile zu einem Programm, das den Hauptspeicher bereits restlos füllt, führt zu der Fehlermeldung

OUT OF MEMORY ERROR(Hauptspeicher nicht ausreichend)

wobei die Zeile nicht mehr hinzugefügt wird.

Bei der Eingabe von Programmzeilen erfolgt keine Syntax–Prüfung. Erst wenn die Zeile vom Interpreter ausgeführt wird, wird die Syntax überprüft und ggf. eine Fehlermeldung (s. Kapitel 8.1) angezeigt.

Bei Eingabe einer ungeraden Anzahl von Anführungszeichen (") wird der Anführungs–Modus eingeschaltet, bei dem wie beim Einfügungsmodus (s. INST–Taste in Kapitel 3.1) der Code einer ggfs. dann gedrückten Cursor-Steuertaste als inverses Zeichen auf dem Bildschirm dargestellt wird. Auf diese Weise können die Cursor–Steuerfunktionen mit Hilfe von Steuerzeichenketten in Verbindung mit der PRINT–Anweisung programmiert werden.

3.3 Ersetzen oder Ändern von existierenden Programmzeilen

Wie bereits in Kapitel 3.2 erwähnt, werden existierende Programmzeilen ersetzt, indem eine neue Programmzeile mit der Zeilennummer der zu ersetzenden Zeile eingegeben wird. Dabei kann die neue Programmzeile auch kürzer oder länger als die zu ersetzende Zeile sein.

Eine Programmzeile wird gelöscht, indem nur die Zeilennummer eingegeben und dann die RETURN–Taste gedrückt wird. Existiert die zu löschende Zeile nicht, so wird die Eingabe ohne Anzeigen einer Fehlermeldung ignoriert.

Das ganze im Arbeitsspeicher befindliche BASIC–Programm kann mit dem Befehl NEW (s. dort in Kapitel 5.2) gelöscht werden.

3.3.1 Ändern von Zeilen mit dem Bildschirmeditor

Jede auf dem Bildschirm sichtbare Zeile kann mit Hilfe des Bildschirmeditors, der immer dann aktiv ist, wenn BASIC sich in der Befehlsebene (Direktmodus) befindet oder ein Programm auf eine Tastatureingabe wartet, geändert werden.

Dazu wird der Cursor (Positionsanzeiger) mit Hilfe der Cursor–Steuertasten (s. Kapitel 3.1) auf die gewünschte Position in der zu ändernden Zeile gesetzt, und dann wird die Änderung – ggf. auch unter Verwendung der INST/DEL–Taste – ausgeführt. Durch Drücken der RETURN–Taste wird die Änderung beendet, und die gesamte geänderte Programmzeile wird ins Programm eingefügt.

Sollen auf dem Bildschirm nicht sichtbare Programmzeilen geändert werden, so werden diese zunächst mit dem LIST– Befehl (s. dort in Kapitel 5.2) auf dem Bildschirm angezeigt und dann wie oben beschrieben geändert.

Die RETURN–Taste, die die Änderung einer Zeile beendet, kann bei jeder beliebigen Cursor–Position innerhalb der geänderten Zeile gedrückt werden. Der Interpreter übernimmt in jedem Fall die gesamte Programmzeile ins Programm.

3.3.2 Ändern von Zeilen mit Syntax–Fehler

Erkennt der Interpreter während der Programmausführung einen Syntax-Fehler, so meldet er dies unter Angabe der Zeilennummer, falls das Pro-

gramm keine programmierte Fehlerverarbeitung enthält. Die fehlerhafte Zeile muß mit LIST oder durch Drücken der HELP–Taste angezeigt und dann mit den oben beschriebenen Techniken geändert werden. Drücken der RETURN–Taste speichert die geänderte Zeile ins Programm, das dann erneut gestartet werden kann.

Bei einer solchen Programmänderung müssen folgende Maßnahmen des Interpreters berücksichtigt werden:

- Alle Variablen und Felder werden gelöscht (Wert oder Länge Null).
- Das Programm kann **nicht** mit dem CONT–Befehl (s. dort in Kapitel 5.2) fortgesetzt werden.

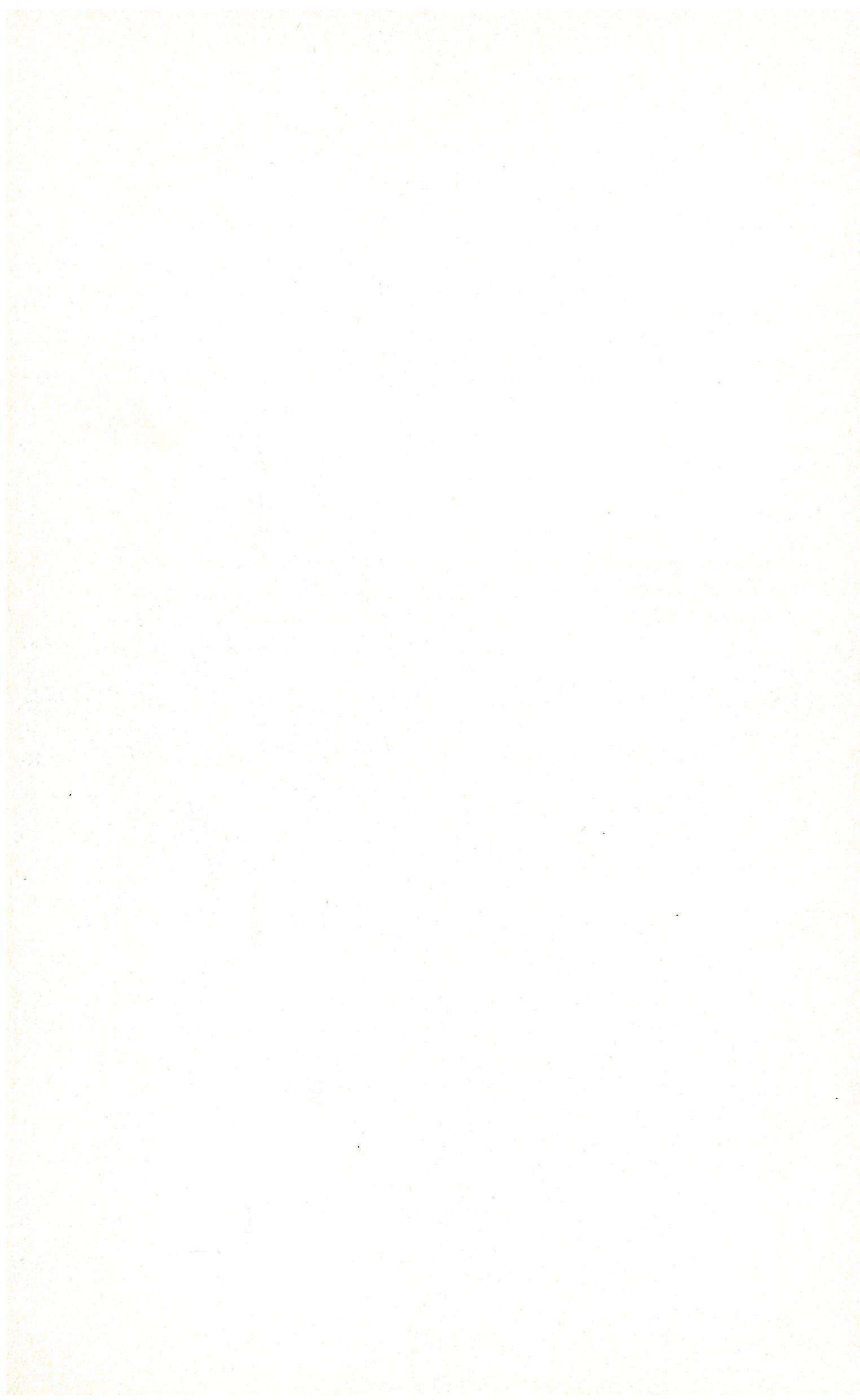
Sollen vor der Änderung die Inhalte bestimmter Variablen oder Felder überprüft werden, muß der Interpreter durch Drücken der Taste RUN/STOP in die Befehlsebene (Direkt modus) gesetzt werden. Dann kann die Überprüfung durch Anzeigen der betreffenden Variablen oder Felder mit PRINT erfolgen, die Zeile geändert und das Programm neu gestartet werden.

3.4 Speichern von BASIC–Programmen

Ein neu eingegebenes oder ein geändertes Programm muß für den Erhalt auf Kassette oder Diskette gespeichert werden. Diese Maßnahme wird mit dem SAVE– oder dem DSAVE–Befehl durchgeführt. Beide Befehle sind in den Kapiteln 5.2 und 4.6 so ausführlich beschrieben, daß hier eine weitere Erläuterung entfallen kann.

3.5 Laden und Starten von BASIC–Programmen

Um ein auf Kassette oder Diskette gespeichertes BASIC–Programm auszuführen, muß dieses zunächst in den Hauptspeicher des Rechners geladen werden. Dies geschieht im Direktmodus des Rechners mit den Befehlen LOAD, DLOAD oder RUN. Nach dem Laden mit LOAD oder DLOAD wird das Programm mit dem Befehl RUN gestartet. Die Befehle LOAD, DLOAD und RUN sind in den Kapiteln 5.2 bzw. 4.6 so ausführlich beschrieben, daß hier auf eine weitergehende Erläuterung verzichtet werden kann.



4. C128-Modus



4. C128–Modus

Der C128 meldet sich in diesem Modus, wenn der Rechner eingeschaltet oder der Reset–Knopf gedrückt wird.

In diesem Modus stehen dem Anwender 128 KBytes Hauptspeicher, wahlweise eine 40 oder 80 Zeichen breite Bildschirmdarstellung mit 25 Zeilen, ein BASIC–Interpreter mit gegenüber dem C64–Modus (s. Kapitel 5) erheblich erweitertem Befehlsumfang (BASIC 7.0), wahlweise eine deutsche DIN– oder eine amerikanische ASCII–Tastatur mit speziellen Bedienungs– und programmierbaren Funktionstasten sowie mit der Diskettenstation 1570/71 ein erheblich schnellerer Diskettenzugriff zur Verfügung.

4.1 Erweiterte Tastatur–Funktionen im C128–Modus

Die in Kapitel 3 beschriebene und abgebildete Tastatur enthält eine Reihe von Tasten, denen nur im C128–Modus eine Funktion zugeordnet ist.

Zehnertastatur: Rechts neben der Schreibmaschinentastatur befindet sich ein Block mit den zehn Zifferntasten, dem Dezimalpunkt, der Plus– und der Minus–Taste sowie einer Eingabetaste. Die Tasten haben dieselbe Funktion wie die entsprechenden Tasten der Schreibmaschinentastatur. Die ENTER-Taste entspricht dabei der RETURN–Taste. Dieser Tastenblock erleichtert die Eingabe großer Zahlenmengen und erlaubt mit Hilfe eines geeigneten Programms die Simulation eines Taschenrechners.

Funktionstasten: Oberhalb der Zehnertastatur befinden sich 4 Funktionstasten für insgesamt 8 verschiedene programmierbare Funktionen. Die Funktionen f1, f3, f5 und f7 werden durch Drücken der entsprechenden Taste und die Funktionen f2, f4, f6 und f8 durch zusätzliches Niederhalten der SHIFT–Taste ausgeführt. Die Funktionen bestehen in simulierten Tastatureingaben und ggfs. dem Ausführen ganzer Befehle oder Befehlsketten.

Nach dem Einschalten des Rechners sind den 8 Funktionstasten folgende Befehle zugeordnet:

f1	GRAPHIC	f2	DLOAD"
f3	DIRECTORY <RETURN>	f4	SCNCLR <RETURN>
f5	DSAVE"	f6	RUN <RETURN>
f7	LIST <RETURN>	f8	MONITOR <RETURN>

Wird **f1** gedrückt, so wird das Befehlswort GRAPHIC bei der aktuellen Cursor–Position angezeigt, und es brauchen nur noch die gewünschten Parameter (s. dort in Kapitel 4.6) eingegeben zu werden.

Wird **f2** (**f1** zusammen mit **SHIFT**) gedrückt, so wird das Befehlswort DLOAD" bei der aktuellen Cursorposition angezeigt, und es braucht nur noch der gewünschte Dateiname mit abschließendem Anführungszeichen eingegeben zu werden.

Wird **f3** gedrückt, so wird der Befehl DIRECTORY ausgeführt, der das Diskettenverzeichnis der Diskette in einem angeschlossenen Floppy–Disk-Laufwerk mit der Geräteadresse 8 anzeigt.

Wird **f4** (**f3** zusammen mit **SHIFT**) gedrückt, so wird der aktuelle Bildschirm mit dem Befehl SCNCLR gelöscht.

Wird **f5** gedrückt, so wird das Befehlswort DSAVE" bei der aktuellen Cursorposition angezeigt, und es braucht nur noch der gewünschte Dateiname mit abschließendem Anführungszeichen eingegeben zu erden.

Wird **f6** (**f5** zusammen mit **SHIFT**) gedrückt, so wird der Befehl RUN ausgeführt, also das gerade im Hauptspeicher befindliche BASIC–Programm gestartet.

Wird **f7** gedrückt, so wird der Befehl LIST ausgeführt, also das gerade im Hauptspeicher befindliche BASIC–Programm auf dem Bildschirm gelistet.

Wird **f8** (**f7** zusammen mit **SHIFT**) gedrückt, so wird der Maschinensprache-monitor mit dem Befehl MONITOR aufgerufen.

Die voreingestellte Belegung der Funktionstasten kann jederzeit mit dem KEY–Befehl (s. dort in Kapitel 4.6) für jede beliebige Taste geändert oder einfach nur angezeigt werden.

Cursor–Steuertasten: Die vier Tasten mit den Pfeil–Symbolen links neben den Funktionstasten dienen der einfachen Steuerung der Cursor–Bewegungen auf dem Bildschirm. Anders als beim C64–Modus gibt es hier für jede Cursor–Richtung eine separate Taste. Die Richtung wird durch den Pfeil auf der jeweiligen Taste angegeben.

HOME–Taste: Wird die HOME–Taste zweimal aufeinanderfolgend gedrückt, so wird ein ggfs. definiertes Bilschirmfenster (s.a. WINDOW–Anweisung in

Kapitel 4.6 sowie die Tastensequenzen ESC T und ESC B in diesem Abschnitt) aufgehoben und auf den gesamten Bildschirm zurückgesetzt.

NO SCROLL–Taste: Wenn während der Anzeige von Daten auf dem Bildschirm dieser nach oben gerollt wird, so kann dies durch Drücken der NO SCROLL–Taste, die in dem Tastenblock links neben den Cursor–Steuertasten ganz rechts angeordnet ist, verhindert werden. Die Funktion dieser Taste ähnelt der der SHIFT LOCK–Taste. Einmaliges Drücken verriegelt die Funktion (Bildschirmrollen unterdrückt). Nochmaliges Drücken entriegelt die Funktion (Bildschirmrollen erlaubt).

40/80 DISPLAY–Taste: Mit Hilfe dieser Taste kann zwischen den beiden Anzeigebreiten des C128–Modus hin– und hergeschaltet werden (gedrückt: 80–Zeichen–Anzeige). Voraussetzung ist jedoch, daß ein 80–Zeichen–Monitor (z.B. Commodore 1902) angeschlossen ist. Auch diese Taste verriegelt sich beim Drücken automatisch und wird bei erneutem Drücken wieder freigegeben. Die Umschaltung muß jedoch **vor** dem Einschalten oder Rücksetzen (Reset–Taste) des Computers erfolgen.

LINE FEED–Taste: Drücken dieser Taste setzt den Cursor auf dieselbe Spalte der folgenden Zeile.

HELP–Taste: Findet der Interpreter während der Abarbeitung einer Programmzeile einen Fehler, so meldet er dies mit einer Fehleranzeige (s. Kapitel 8.1). Wird nun die HELP–Taste gedrückt, so wird die fehlerhafte Zeile angezeigt und die Zeile wird ab der fehlerhaften Stelle je nach Bildschirmmodus entweder invers (40–Zeichen–Bildschirm) oder unterstrichen (80–Zeichen–Bildschirm) dargestellt. Auf diese Weise wird die Programmkorrektur sehr erleichtert.

ASCII/DIN–Taste: Dies ist eine selbstverriegelnde Taste, mit der zwischen der deutschen DIN– und der amerikanischen ASCII–Tastatur umgeschaltet werden kann. Bei gedrückter (und damit verriegelter) Taste ist die deutsche DIN–Tastatur aktiviert.

ALT–Taste: Mit dieser Taste kann eine alternative, selbstdefinierte Tastaturtabelle gewählt werden, in der fast allen Tasten der Schreibmaschinentastatur ein anderer Code zugeordnet werden kann. Dies ist z.B. dann hilfreich, wenn Texte in einer fremden Sprache geschrieben werden müssen. Die Definition solcher Tabellen erfolgt durch Speichern von bestimmten Werten in besonderen Speicherplätzen (s. Anhang J).

TAB–Taste: Drücken der TAB–Taste zusammen mit der SHIFT–Taste setzt an der aktuellen Cursor–Position einen Tabulator–Stop, der durch Drücken der TAB–Taste angesprungen werden kann. Voreingestellt sind Tabulatorstops alle 8 Spalten.

ESC–Taste: Die ESC–Taste erlaubt zusammen mit den Buchstabentasten A bis Z eine Reihe von Sonderfunktionen für die Tastatur und den Bildschirm, die im folgenden beschrieben werden. Zur Ausführung der gewünschten Funktion wird zuerst die ESC–Taste gedrückt und wieder losgelassen. Anschließend wird die gewünschte Buchstabentaste gedrückt:

Taste	Funktion
A	Der automatische Einfügemodus wird eingeschaltet. Es werden solange Zeichen eingefügt, bis ESC–C gedrückt wird.
B	Die aktuelle Cursor–Position definiert die untere rechte Ecke eines Bildschirmfensters.
C	Der automatische Einfügemodus wird ausgeschaltet.
D	Die aktuelle Bildschirmzeile wird gelöscht und der Bildschirm unterhalb dieser Zeile wird um eine Zeile nach oben gerollt.
E	Setzt den Cursor–Modus von Blinken auf Konstantanzeige.
F	Setzt den Cursor–Modus von Konstantanzeige auf Blinken.
G	Erlaubt akustisches Klingeln durch CTRL–G.
H	Verbietet akustisches Klingeln durch CTRL–G.
I	Fügt oberhalb der aktuellen Zeile eine neue Zeile auf dem Bildschirm ein.
J	Setzt den Cursor an den Anfang der aktuellen Zeile.
K	Setzt den Cursor hinter das letzte Zeichen in der aktuellen Zeile.
L	Erlaubt Bildschirm–Aufrollen.
M	Verbietet Bildschirm–Aufrollen.

ESC–Sequenzen (Fortsetzung)

Taste	Funktion
N	Schaltet den Bildschirm von Invers–Darstellung auf Normal–Darstellung (nur 80–Zeichen–Schirm).
O	Bewirkt ein Abschalten aller Modi (Blinken, Untersteichen, Revers).
P	Die aktuelle Zeile wird vom Anfang bis zur Cursor–Position gelöscht.
Q	Die aktuelle Zeile wird von der Cursor–Position bis zum Zeilenende gelöscht.
R	Schaltet den Bildschirm von Normal–Darstellung auf Invers–Darstellung (nur 80–Zeichen–Schirm).
S	Schaltet Blockdarstellung für den Cursor ein.
T	Die aktuelle Cursor–Position definiert die obere linke Ecke eines Bildschirmfensters.
U	Schaltet Strichdarstellung für den Cursor ein (nur 80–Zeichen-Bildschirm).
V	Rollt den Bildschirm um eine Zeile nach oben.
W	Rollt den Bildschirm um eine Zeile nach unten.
X	Schaltet von einem 40–Zeichen–Monitor auf einen 80–Zeichen-Monitor um und umgekehrt.
Y	Schaltet voreingestellte Tabulatorstops (alle 8 Spalten) ein.
Z	Löscht alle eingestellten Tabulatorstops.
@	Löscht den Bildschirm ab der aktuellen Cursor–Position.

CTRL–Taste Zusätzlich zu den Funktionen, die mit speziellen CTRL–Sequenzen im C64–Modus ausgelöst werden können (z.B. Farbwahl), erlaubt der C128–Modus eine Reihe von weiteren Funktionen, die durch Drücken

einer der folgenden Tasten bei niedergehaltener CTRL–Taste ausgelöst werden können:

Taste	CHR$–Code	Funktion
B	2	Schaltet den Unterstreichungsmodus für alle angezeigten Zeichen ein (nur 80–Zeichen–Bildschirm)
G	7	Löst ein akustisches Klingelzeichen aus.
I	9	Löst einen Tabulatorsprung auf den nächsten Tabulatorstop aus.
J	10	Führt einen Zeilenvorschub aus.
K	11	Verbietet Umschaltung zwischen Klein/ Groß– und Groß/Grafik–Zeichensatz.
L	12	Erlaubt Umschaltung zwischen Klein/ Groß– und Groß/Grafik–Zeichensatz.
O	15	Schaltet Blinkdarstellung für alle angezeigten Zeichen ein (nur 80–Zeichen-Bildschirm).
X	24	Setzt oder löscht einen Tabulator–Stop.
[	27	ESC–Code.

Achtung
Die ESC– und CTRL–Funktionen können auch mit Hilfe der PRINT–Anweisung in einem Programm ausgeführt werden. Z.B.:

PRINT CHR$(27) + "M" verbietet Bildschirmrollen.

PRINT CHR(7) löst ein akustisches Klingelzeichen aus.

Bei aktivierter DIN–Tastatur gilt für die Verwendung der Tasten zusammen mit der CTRL–Taste die ASCII–Tastaturbelegung.

Besonderheiten der DIN/ASCII–Tastatur

Anwender, die eine reine amerikanische Tastatur einsetzen wollen, können mit dem Befehl

`POKE 2757,129` [nur im 80–Zeichen–Modus]

die Unterstützung der Tastaturumschaltung durch das Betriebssystem ausschalten. Diese Maßnahme sollte jedoch nur bei vorher aktivierter ASCII–Tastatur (freigegebene ASCII/DIN–Taste) ergriffen werden und kann nur durch Drücken des Reset–Knopfes wieder aufgehoben werden.

Eine weitere Besonderheit der DIN–Tastatur stellt die Taste mit den beiden Accents in der obersten Reihe der Schreibmaschinentastatur dar. Wird diese Taste gedrückt, so wird der Accent angezeigt, der Cursor bleibt aber bei dieser Position stehen. Wird dann eine Buchstabentaste gedrückt, so prüft das Tastaturprogramm des Betriebssystems, ob diese Zeichenkombination erlaubt ist. Wenn nicht, wird nur der Buchstabe, andernfalls der akzentuierte Buchstabe angezeigt und der Cursor wird um eine Position nach rechts gesetzt.

Damit Sie sich bei jeder aktivierten Tastatur auch zurechtfinden, sind auf einigen Tasten, die bei den beiden wählbaren Tastaturen unterschiedlich belegt sind, sämtliche Belegungen vermerkt.

4.2 Erweiterte Bildschirm–Funktionen im C128–Modus

Im C128–Modus kann der Anwender zwischen einer 40– und einer 80–Zeichen–Bildschirmanzeige wählen. Beide Anzeigemodi können in einem Programm verwendet werden.

Die 40–Zeichen–Anzeige ist auch die, die für den C64–Modus gültig ist. Außerdem dient sie zur Darstellung all der vielfältigen grafischen Möglichkeiten, die der C128–Modus bietet (s.a. Kapitel 4.7 ff.).

Die 80–Zeichen–Anzeige erlaubt die Ausführung professioneller Programme wie Textverarbeitung, Datenbankverwaltung usw. Hochauflösende Grafik ist bei dieser Darstellung nur in Verbindung mit PEEK– und POKE–Befehlen oder einem Maschinespracheprogramm möglich. Die Verwendung der grafischen Sonderzeichen aus dem Commodore–Zeichensatz sowie der Hintergrund– und Vordergrundfarben sind ebenfalls möglich.

40/80 DISPLAY–Taste Mit dieser Taste kann zwischen der 40– und der 80–Zeichen–Anzeige umgeschaltet werden. Die Taste kann zu jeder Zeit gedrückt werden. Die Umschaltung erfolgt jedoch erst dann, wenn

- der Rechner aus– und wieder eingeschaltet wird;
- die Reset–Taste neben dem Netzschalter gedrückt wird;
- die RUN/STOP–Taste mit der RESTORE–Taste gedrückt wird.

Diese Taste verriegelt sich wie die SHIFT LOCK–Taste beim Drücken selbsttätig und wird durch erneutes Drücken wieder entriegelt. Im gedrückten Zustand ist die 80–Zeichen–, andernfalls die 40–Zeichen–Anzeige eingeschaltet.

Video–Anschlüsse und Monitore Wie bereits in der Einführung, hat der C128–Computer auf der Rückseite zwei unterschiedliche Video–Anschlüsse (VIDEO und RGBI). An den Anschluß VIDEO wird ein 40–Zeichen-Monitor (z.B. Commodore 1701) angeschlossen.

An den Anschluß RGBI wird ein 80–Zeichen–Monitor angeschlossen (RGBI–Monitor).

Beide Anschlüsse werden benötigt, wenn ein Doppel–Monitor wie der Commodore 1901 angeschlossen wird. Dieser Monitor kann sowohl im 40–

als auch im 80–Zeichen–Modus betrieben werden und erlaubt die programmierte Umschaltung zwischen beiden Modi.

Werden zwei verschiedene Monitore angeschlossen, so können sowohl grafische– als auch Textausgaben gleichzeitig dargestellt werden.

Bildschirmumschaltung Einer der großen Vorteile des C128–Modus ist die Möglichkeit, mit bis zu 3 voneinander unabhängigen Bildschirmen zu arbeiten (80–Zeichen–Bildschirm sowie geteilter Bildschirm für grafische und Textwiedergabe). Es kann auf einem Bildschirm ausgegeben werden, ohne die auf den beiden anderen Schirmen angezeigten Daten zu verändern und es kann zwischen allen drei Bildschirmen ohne Datenverlust hin– und hergeschaltet werden. Außerdem kann zwischen der 40– und der 80–Zeichen–Anzeige umgeschaltet werden, ohne daß der Bildschirm vorher gelöscht werden muß. Diese Umschaltung wird durch aufeinanderfolgendes Drücken der Tasten **ESC** und **X** oder durch **PRINT CHR$(27)** + "X" bewirkt.

Zur programmierten Bildschirmumschaltung dient die GRAPHIC–Anweisung (s. dort in Kapitel 4.6). Mit ihr lassen sich in Programmen für den 80–Zeichen–Modus grafische Darstellung auf dem 40–Zeichen–Bildschirm anzeigen.

Weitere Informationen siehe Kapitel 4.6 und 4.7 ff.

4.3 BASIC 7.0 im C128–Modus

Aufbauend auf dem Commodore–Standard–BASIC 2.0, das ausführlich im Kapitel 5.2 beschrieben wird, bietet das erweiterte BASIC 7.0 dem Anwender viele komfortable Befehle, Anweisungen, Funktionen und Systemvariablen, um die vielfältigen Möglichkeiten, die der C128 bietet, nutzen zu können.

Dieser Abschnitt ist kein Leitfaden zum Erlernen allgemeiner Programmiertechniken oder der Programmiersprache BASIC, sondern ein Nachschlagewerk für mit allgemeinen Programmierkonzepten vertraute Anwender, die auf den nachfolgenden Seiten alle wissenswerten Fakten für die Erstellung effektiver BASIC–Programme finden. Anfängern wird die Lektüre der in der Commodore–Sachbuchreihe veröffentlichten BASIC–Kurse empfohlen.

Im C128–Modus kann der volle Befehlsumfang sowohl des BASIC 2.0 (s. Kapitel 5) als auch des BASIC 7.0 verwendet werden, so daß hier auch auf das Kapitel 5.2 verwiesen wird.

Ehe jedoch die einzelnen Anweisungen beschrieben werden, hier noch einige allgemeine Anmerkungen zur Struktur der Programmiersprache BASIC und zur Syntax der nachfolgenden Beschreibungen.

4.4 Struktur von BASIC

Der Sprachumfang des Commodore–BASIC gliedert sich in:

- Befehle
- Anweisungen
- Funktionen
- Variablen

Befehle werden dazu benutzt, an Programmen etwas zu bearbeiten, sie zu listen, zu ändern, zu löschen, zu speichern oder zu laden. Deshalb werden Befehle auch meistens in der Befehlsebene (Direktmodus) eingegeben. Sie können aber auch Bestandteil eines Programms sein.

Anweisungen steuern den Programmablauf. Sie sind der eigentliche Inhalt des Programms, können aber auch im Direktmodus eingegeben werden. Die Anweisungen gliedern sich in 3 Gruppen:

- Ein–/Ausgabeanweisungen für den Datenverkehr zwischen dem Rechner einerseits sowie Tastatur, Bildschirm, Diskettenstation, Festplatte, Drucker oder Datenfernübertragungsschnittstelle andererseits.

- Deklarationsanweisungen für die Definition von Daten, benutzereigenen Funktionen, Speichersegmenten oder der Programmumgebung.

- Steueranweisungen zur Festlegung des Programmablaufes wie Schleifen, Sprünge, Unterprogramm– und Unterbrechungsaufrufe.

Funktionen liefern als Ergebnis bestimmte Werte zur Weiterverarbeitung an das Programm und gliedern sich in:

- Numerische Funktionen, die als Ergebnis ganzzahlige Werte oder Gleitkommawerte übergeben;

- Zeichenkettenfunktionen, die als Ergebnis eine Zeichenkette übergeben.

Variablen der BASIC–Sprache dürfen nicht mit den vom Programmmierer frei definierbaren Variablen eines Programms verwechselt werden. Die hier gemeinten System–Variablen haben feste, von BASIC vorgegebene Namen und Funktionen. Ihnen wird beim Aufruf vom Interpreter ein Wert zugewiesen, der bestimmte Funktionszustände des Interpreters oder Rechnersystems zu einem bestimmten Zeitpunkt beschreibt. Es gibt numerische und Zeichenketten–Systemvariablen.

4.5 Struktur und Syntax der Beschreibungen

Im Kapitel 4.6 werden alle Befehle, Anweisungen, Funktionen und Variablen, die der BASIC–Interpreter kennt, detailliert und an Hand von Beispielen beschrieben. Die einzelnen Beschreibungen sind alphabetisch nach den Namen geordnet und nach dem folgenden Schema gegliedert:

Format: Hier ist die exakte Schreibweise für einen Befehl, eine Anweisung, Funktion oder Variable angegeben. Dabei gelten die folgenden Syntax–Regeln:

1. Alle Wörter in Großbuchstaben sind BASIC–Schlüsselwörter, deren Schreibweise wie angegeben verbindlich ist.

2. Alle Angaben in kursiver Groß– und Kleinschrift stellen Parameter dar, die vom Anwender eingesetzt werden müssen.

3. Angaben zwischen eckigen Klammern ([]) sind wahlfrei.

4. Eine Folge von Punkten (. . .) hinter einer Angabe bedeutet, daß diese Angabe, falls erforderlich, mehrmals spezifiziert werden kann.

5. Alle angegebenen Sonderzeichen außer den eckigen Klammern, also , ; – = ( ) ' ", müssen da, wo sie stehen, auch angegeben werden.

6. Bei den Parameterangaben haben die dort verwendeten Abkürzungen folgende Bedeutung:

x,y,z	beliebige numerische Ausdrücke
i,j,k,m,n	beliebige ganzzahlige Ausdrücke
x$,y$	beliebige Zeichenkettenausdrücke
v,v$	numerische oder Zeichenkettenvariablen

Zweck: Hier wird die Wirkung des jeweiligen Befehls oder der Anweisung, Funktion oder Variablen detailliert beschrieben, und die Parameter und ihre Definitionsbereiche werden erläutert.

Bemerkungen: Besondere Eigenschaften der einzelnen Befehle, Anweisungen, Funktionen und Variablen werden hier erläutert.

Beispiel: Die bei jeder Beschreibung angegebenen Beispiele zeigen, wie die einzelnen Befehle, Anweisungen, Funktionen und Variablen angewendet werden und weisen auf Besonderheiten hin.

4.6 Beschreibung der einzelnen Befehle, Anweisungen, Funktionen und Variablen

Zur Erleichterung des Auffindens eines bestimmten BASIC–Befehls oder einer –Anweisung, –Funktion oder –Variablen sind die nachfolgenden Beschreibungen nicht entsprechend der in Kapitel 4.4 beschriebenen Sprachstruktur von BASIC gegliedert, sondern es sind alle Sprachelemente zusammengefaßt und alphabetisch geordnet.

APPEND–Anweisung

Format: **APPEND #*log Filnr,Dateiname*[,D*Laufw*]**
[,U*Geräteadr*]

Zweck: Öffnet eine existierende, sequentielle Datei vom Typ SEQ oder USR als Ausgabedatei und positioniert den Schreibzeiger auf das Dateiende, damit weitere Daten angefügt werden können (s.a. Kapitel 6.6).

log Filenr Ein ganzzahliger Wert zwischen 1 und 255, der der eröffneten Datei als Kennummer zugeordnet wird. Bei Werten kleiner als 128 wird an die mit der PRINT#–Anweisung ausgegebenen Daten ein Wagenrücklaufcode ausgegeben. Bei Werten über 127 wird zusätzlich ein Zeilenvorschubcode ausgegeben.

Dateiname Eine Zeichenkettenkonstante in Anführungsstrichen (") oder –variable von maximal 16 Zeichen Länge für den Namen der zu eröffnenden Datei.

Laufw Ein ganzzahliger Wert von 0 oder 1. Voreingestellt ist hier 0.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15 für die Geräteadresse der angeschlossenen Floppy–Disk. Voreingestellt ist hier 8.

Bemerkungen: Werden für ***Dateiname, Laufwerk*** und ***Geräteadr*** Variablen verwendet, so müssen diese in Klammern gesetzt werden.

Beispiel 1:

```
APPEND #2,"TEXT"
```

Auf der Floppy–Disk mit der Geräteadresse 8 wird die Datei TEXT unter der logischen File–Nummer 2 eröffnet und der Schreibzeiger wird auf das Ende der Datei gestellt, so daß weitere Daten angefügt werden können.

Beispiel 2:

```
APPEND #3,(DA$),D(VAL(AL$)) ON U9
```

Der Dateiname und das Laufwerk werden als Variable übergeben.

AUTO–Befehl

Format: **AUTO** [***Schrittw***]

Zweck: Während der Eingabe von BASIC–Programmzeilen wird am Zeilenanfang eine neue Zeilennummer angezeigt, wenn am Ende der vorhergehenden Zeile die RETURN-Taste gedrückt wurde.

Nach Eingabe des Befehls AUTO muß die erste Programmzeile mit Zeilennummer eingegeben werden. Ab der nächsten Zeile erfolgt die Zeilennumerierung automatisch.

Bemerkungen: ***Schrittw*** gibt die Schrittweite zwischen den Zeilennummern an.

Wird der AUTO–Befehl ohne ***Schrittw*** verwendet, so wird die automatische Zeilennumerierung wieder abgeschaltet.

Beispiele:

```
AUTO 10
```

Es werden automatisch Zeilennummern mit einer Schrittweite von 10 erzeugt.

```
AUTO
```

Die automatische Zeilennummernerzeugung während der Programmeingabe wird abgeschaltet.

BACKUP–Befehl

Format: **BACKUP D*Quelle* TO D*Ziel*[,U*Gerät*]**

Zweck: Dupliziert eine Diskette auf einem Floppy–Disk–Doppellaufwerk.

Quelle Ein Wert von 0 oder 1 für das Laufwerk, in dem die zu duplizierende Diskette eingelegt ist.

Ziel Ein Wert von 1 oder 0 für das Laufwerk, in dem die Diskette auf die kopiert wird, eingelegt ist.

Gerät Ein ganzzahliger Wert zwischen 4 und 15 für die Geräteadresse des angeschlossenen Doppellaufwerkes. Voreingestellt ist hier 8.

Bemerkungen: Der BACKUP–Befehl ist nur bei einer Floppy–Disk mit zwei Laufwerken möglich.

Das Duplikat ist ein exaktes Abbild der Quelldiskette.

Der BACKUP–Befehl beinhaltet den Befehl HEADER (s. dort), mit dem die Diskette vor dem Duplizirvorgang formatiert wird.

Vor dem Beginn des Duplizierorganges wird durch die Anfrage

```
ARE YOU SURE?
```

vom Anwender eine Bestätigung (Anwort Y für YES) verlangt, da durch BACKUP ggfs. vorher auf der Zieldiskette vorhandene Daten zerstört werden.

Beispiel 1:

```
BACKUP D0 TO D1
```

Duplizierung der Diskette in Laufwerk 0 auf die Diskette in Laufwerk 1.

Beispiel 2:

```
BACKUP D1 TO D0 ON U9
```

Duplizierung von Laufwerk 1 auf Laufwerk 0 im Gerät mit der Adresse 9.

BANK–Befehl

Format: **BANK *n***

Zweck: Definiert eine 64–kByte–Speicherbank für nackfolgende PEEK–, POKE–, SYS– oder WAIT–Anweisungen. *n* muß zwischen 0 und 15 liegen.

Bemerkungen: Der Speicher des C128 ist in 15 Speicherbänke zu je 64 kBytes aufgeteilt. Dabei sind jedoch nur die Bänke Nr. 0, 1 und 15 vorhanden (s.a. Speicherverwaltung in Anhang B).

Beispiel:

```
10 REM WERT 20 BEI ADRESSE 1024 IN BANK
15 REM 1 SPEICHERN
20 BANK 1:POKE 1024,20
```

BEGIN– und BEND–Anweisungen

Format:

IF . . . THEN BEGIN

.

. Anweisungen

.

BEND

Zweck:

BEGIN und BEND schließen einen beliebig langen Block von BASIC–Anweisungen ein, der auch über mehrere Zeilen gehen darf.

Bemerkungen:

BEGIN wird immer in Verbindung mit der IF–Anweisung (s. dort) verwendet.

Der Vorteil gegenüber der IF THEN ELSE–Anweisung ist der, daß die Anweisungen nach THEN nicht auf eine Programmzeile beschränkt zu sein brauchen.

Alle anderen Regeln für die IF. . .THEN. . .ELSE–Anweisung bleiben jedoch gültig (s. Beispiel).

Beispiel 1:

```
100 IF X=1 THEN BEGIN:A=5
110 B=6:C=7
120 PRINT A+B+C:BEND:PRINT "AHA !"
130 . . .
```

AHA ! wird nur gedruckt, wenn X = 1 logisch wahr ist; andernfalls würde das Programm mit Zeile 130 fortgesetzt.

Beispiel 2:

```
10 INPUT "ZAHL=";Z
20 IF Z>0 THEN BEGIN
30 PRINT "DIE ZAHL IST > 0"
40 BEND:ELSE GOTO 60
50 END
60 PRINT "DIE ZAHL IST <=0"
70 GOTO 10
```

Das Programm wird in Zeile 50 beendet, wenn eine positive Zahl eingegeben wurde, andernfalls mit Zeile 10 wiederholt.

BLOAD–Befehl

Format: **BLOAD** ***Dateiname***[**,D*Laufwerk***][**,U*Gerät***]
[**,ON B*Bank***][**,P*Adr***]

Zweck: Es wird eine Datei mit binär codiertem Inhalt (z.B. Maschinensprache, Binärdaten) in den Hauptspeicher geladen (s.a. Kapitel 6.4).

Dateiname Eine Zeichenkettenkonstante in Anführungszeichen (") oder –variable einer Länge von bis zu 16 Zeichen, die den Namen der zu ladenden Binärdatei angibt.

Laufwerk Ein Wert von 0 oder 1 für das Laufwerk, von dem geladen werden soll. Voreingestellt ist hier 0.

Gerät Ein ganzzahliger Wert zwischen 4 und 15 für die Geräteadresse der angeschlossenen Floppy–Disk.

Bank Ein ganzzahliger Wert zwischen 0 und 15 für die Speicherbank, in die geladen werden soll. Voreingestellt ist hier 0.

Adr Ein ganzzahliger Wert zwischen 0 und 65535, der die Adresse in der gewählten Bank angibt, ab der geladen werden soll. Voreingestellt ist hier die Adresse, die sich aus den ersten beiden Bytes der Binärdatei ergibt.

Bemerkungen: Werden für ***Dateiname, Laufwerk, Gerät, Bank*** und/oder ***Adr*** Variablen verwendet, so müssen diese in Klammern angegeben werden.

Beispiel 1:

```
BLOAD"BIN",ON B1,P1024
```

Die Datei BIN wird von der Floppy–Disk mit der Geräteadresse 8 in Bank 1 des Hauptspeichers ab Adresse 1024 geladen.

Beispiel 2:

```
BLOAD(D$),D(LW),U(GA),ON B(B),P(AD)
```

In diesem Beispiel werden für alle Parameter Variablen verwendet.

BOOT–Befehl

Format: **BOOT** [***Dateiname***[,**D*Laufw***],[**U*Gerät***]]

Zweck: Lädt einen ausführbaren Binärfile von Diskette in den Hauptspeicher und startet ihn bei seiner Startadresse.

Dateiname Eine Zeichenkettenkonstante in Anführungszeichen (") oder –variable einer Länge von bis zu 16 Zeichen, die den Namen der zu ladenden Binärdatei angibt.

Laufwerk Ein Wert von 0 oder 1 für das Laufwerk, von dem geladen werden soll. Voreingestellt ist hier 0.

Gerät Ein ganzzahliger Wert zwischen 4 und 15 für die Geräteadresse der angeschlossenen Floppy–Disk.

Bemerkungen: Werden als Parameter Variablen verwendet, so müssen diese in Klammern angegeben werden.

Wird BOOT ohne Parameter verwendet, so entnimmt das Betriebssystem dem Sektor 0 in der Spur 1 der Diskette in der Floppy–Disk Nr. 8 die Information, was geladen und gestartet werden soll. Diese Information kann mit Hilfe der Direktzugriffsbefehle auf die Diskette gebracht werden. Näheres siehe Floppy–Disk–Handbuch.

Beispiel:

```
BOOT "HILFE"
```

Von der Floppy–Disk mit der Geräteadresse 8 wird der ausführbare Binärfile HILFE geladen und bei seiner Startadresse gestartet.

BOX–Anweisung

Format: **BOX** [*Farbquelle*],*x1,y1*[,[*x2,y2*]]
[,[*Winkel*]][,[*Ausmal*]]

Zweck: Es wird ein Rechteck an einer wählbaren Bildschirmposition gezeichnet und wahlweise ausgemalt.

Farbquelle Ein ganzzahliger Wert zwischen 0 und 3, der die Quelle für die Farbe definiert, in der gezeichnet wird. Es bedeuten:

0 Hintergrund
1 Vordergrund
2 Zusatzfarbe 1
3 Zusatzfarbe 2

x1,y1 x2,y2 Koordinaten der linken oberen bzw. rechten unteren Ecke des Rechtecks. Die *x*–Koordianten gehen von 0 bis 320 und die *y*–Koordinaten von 0 bis 199. Voreingestellt sind der Punkt links oben im Bildschirm sowie die aktuelle Position des Pixel–Cursors (grafischer Cursor).

Winkel Winkel in Grad, um den das Rechteck um seinen Mittelpunkt im Uhrzeigersinn gedreht werden soll. Voreingestellt ist der Wert 0.

Ausmal Ein ganzzahliger Wert, der festlegt, ob das Rechteck in der gewählten Farbe ausgemalt werden soll (1) oder nicht (0).

Bemerkungen: Nach dem Zeichnen des Rechteckes ist der Punkt ***x2,y2*** die aktuelle Position des grafischen Cursors.

Beispiele:

```
BOX 1,10,10,60,60
```

Zeichnet die Kontur eines Rechteckes in der aktuellen Vordergrundfarbe.

```
BOX ,10,10,60,60,45,1
```

Zeichnet eine in der aktuellen Vordergrundfarbe ausgefüllte Raute.

```
BOX ,30,90,,45,1
```

Zeichnet ein um 45 Grad gedrehtes, ausgefülltes Polygon.

BSAVE–Befehl

Format: **BSAVE *Dateiname*[,D*Laufwerk*][,U*Gerät*
[,ON B*Bank*][P*Adr1*][TO P*Adr2*]**

Zweck: Es wird ein Hauptspeicherbereich (z.B. Maschinensprache, Binärdaten) als Binärdatei auf Diskette gespeichert.

Dateiname Eine Zeichenkettenkonstante in Anführungszeichen (") oder –variable einer Länge von bis zu 16 Zeichen, die den Namen der zu ladenden Binärdatei angibt.

Laufwerk Ein Wert von 0 oder 1 für das Laufwerk, von dem geladen werden soll. Voreingestellt ist hier 0.

Gerät Ein ganzzahliger Wert zwischen 4 und 15 für die Geräteadresse der angeschlossenen Floppy-Disk.

Bank Ein ganzzahliger Wert zwischen 0 und 15 für die Speicherbank, in die geladen werden soll. Voreingestellt ist hier 0.

Adr1, Adr2 Zwei ganzzahlige Werte zwischen 0 und 65535, die die Anfangs– und Endadressen des Speicherbereiches in der gewählten Bank angeben, der gespeichert werden soll.

Bemerkungen: Werden für ***Dateiname, Laufwerk, Gerät, Bank*** und/oder ***Adr1*** und ***Adr2*** Variablen verwendet, so müssen diese in Klammern angegeben werden.

Beispiel 1:

```
BSAVE"BIN",ON B2,P1024 TO P2047
```

Der Hauptspeicherbereich von Adresse 1024 bis 2047 einschließlich (1 kBytes) in Bank 2 wird als Datei BIN auf Diskette in der Floppy–Disk mit der Geräteadresse 8 gespeichert.

Beispiel 2:

```
BSAVE(D$),D(LW),U(GA),ON B(B)
      ,P(A1) TO P(A2)
```

Bei diesem Beispiel werden für alle Parameter Variablen verwendet.

BUMP–Funktion

Format: ***v*****=BUMP(*****n*****)**

Zweck: Liefert die Nummer des Sprites (1 bis 8), das seit der letzten BUMP–Abfrage entweder mit einem anderen Sprite (***n***=1) oder mit angezeigten Daten (***n***=2) kollidiert ist.

Bemerkungen: Wenn BUMP verwendet wird, braucht die Programmunterbrechung für Sprite–Kollisionen nicht mit der COLLISION-Anweisung (s. dort) aktiviert zu sein.

Bei Mehrfachabfragen sollten vorher mit Hilfe der RSPRITE–Funktion die Sprite–Attribute abgefragt werden, um feststellen zu können, welches Sprite womit kollidiert ist.

BUMP(*n*) wird nach einer Abfrage auf Null gesetzt.

Beispiel:

```
100 ON BUMP(2) GOTO 120,130,140,150
```

Wenn Sprite Nr. 1 mit angezeigten Daten kollidiert ist, verzweigt das Programm nach Zeile 120, bei Nr. 2 nach Zeile 130 usw.

CATALOG–Befehl

Der CATALOG–Befehl ist mit dem DIRECTORY–Befehl (s. dort) identisch.

CHAR–Anweisung

Format: **CHAR** [***Farbquelle***],***x,y***[,***Zeichenkette***]
[,***Invers***]

Zweck: Zeigt eine Zeichenkette mit auswählbarer Position auf dem Bildschirm an.

Farbquelle Ein ganzzahliger Wert zwischen 0 und 3, der die Farbe definiert, in der der Text angezeigt werden soll. Es bedeuten:

0	Hintergrundfarbe
1	Vordergrund– (Zeichen–)Farbe
2	Zusatzfarbe 1
3	Zusatzfarbe 2

x,y Die Spalten– (***x***) und die Zeilen (***y***)–Position, ab der der Text angezeigt werden soll. Spalten: 0 bis 39 bzw. 79; Zeilen: 0 bis 24).

Zeichenkette Der anzuzeigende Text in Anführungszeichen (").

Invers Ein Parameter für die inverse Darstellung des Textes. Hat er den Wert 0, wird der Text normal, hat der den Wert 1, invers dargestellt.

Bemerkungen: Im Text–Modus arbeitet die CHAR–Anweisung analog zur PRINT–Anweisung (s. dort in Kapitel 3), d.h. auch eventuelle Cursor–Steuercodes in der Zeichenkette werden ausgeführt.

Im Grafik–Modus werden solche Steuercodes **nicht** ausgeführt.

Um im Mehrfarbenmodus (s. Kapitel 4.7) Zeichen in der Zusatzfarbe 2 anzuzeigen, muß ***Farbquelle*** auf 0 und ***Invers*** auf 1 gesetzt werden.

Um im Mehrfarbenmodus Zeichen in der Zusatzfarbe 1 anzuzeigen, muß ***Farbquelle*** auf 1 und ***Invers*** auf 0 gesetzt werden.

Beispiel:

```
100 CHAR 1,38,12,"GUTEN MORGEN"
```

Der Text "GUTEN MORGEN" wird in der aktuellen Vordergrundfarbe in der Bildschirmmitte angezeigt.

CIRCLE–Anweisung

Format: **CIRCLE** [*Farbquel*],[*x,y*],*xr*[,[*yr*]
[,[*Start*][,[*Ende*][,[*Winkel*]
[,*Segmentwinkel*]

Zweck: Es wird eine Ellipse, ein Kreis oder ein beliebiges Polygon mit dem Mittelpunkt ***x,y*** auf dem Bildschirm gezeichnet.

Farbquel Ein ganzzahliger Wert zwischen 0 und 3, der die Farbe definiert. Es bedeuten:

0	Hintergrundfarbe
1	Vordergrund– (Zeichen–)Farbe
2	Zusatzfarbe 1
3	Zusatzfarbe 2

x,y Die skalierten (s. SCALE–Anweisung) Mittelpunktkoordinaten der Ellipse. Voreingestellt ist hier die aktuelle Position des grafischen Cursors (Pixel-Cursor).

xr Radius der Hauptachse (skaliert).

yr Radius der Nebenachse (skalie rt). Voreingestellt ist hier ***xr***.

Start,Ende Numerische Ausdrücke, deren Werte Winkelangaben darstellen und die angeben, wo das Zeichnen der Figur beginnen und enden soll. Die Winkel werden entgegen der mathematischen Konvention von 0 in Uhrzeigerrichtung gezählt. Voreingestellt sind für ***Start*** 0 und für ***Ende*** 360.

Winkel Numerischer Ausdruck, dessen Wert den Winkel in Grad angibt, um den die Figur im Uhrzeigersinn gedreht werden soll. Voreingestellt ist hier der Wert 0.

Segmentwinkel Numerischer Ausdruck, dessen Wert den Winkel angibt, um den jedes Segment der Figur gegenüber dem vorhergehenden versetzt werden soll. Voreingestellt ist hier der Wert 2.

Bemerkungen:

Nach dem Zeichnen der Figur ist die aktuelle Position des grafischen Cursors der Endpunkt des Winkels ***Ende***.

Jede Drehung der Figur erfolgt grundsätzlich um den Mittelpunkt ***x,y***.

Werden ***xr*** und ***yr*** gleichgesetzt, so wird ein Kreis gezeichnet.

Der Parameter ***Segmentwinkel*** bestimmt die Anzahl der Seiten (Eckigkeit) der Figur. Kleinere Werte ergeben rundere Konturen.

Beispiele:

```
CIRCLE,160,100,65,10
```

Zeichnet eine Ellipse in der aktuellen Vordergrundfarbe.

```
CIRCLE,160,100,65,50
```

Zeichnet einen Kreis.

```
CIRCLE,60,40,20,18,,,,45
```

Zeichnet ein Achteck.

```
CIRCLE,260,40,20,,,,,90
```

Zeichnet eine Raute.

```
CIRCLE,60,140,20,18,,,,120
```

Zeichnet ein Dreieck.

COLLECT–Befehl

Format: **COLLECT [D*Laufwerk*][ON U*Geräteadr*]**

Zweck: Löscht alle nicht geschlossenen Dateien und gibt den solchen Dateien zugewiesenen Speicherplatz auf der Diskette im spezifizierten Laufwerk wieder frei (s.a. Kapitel 6.9.1).

Laufwerk Ganzzahliger Wert von 0 oder 1. Voreingestellt ist hier 0.

Geräteadr Ganzzahliger Wert zwischen 8 und 15. Voreingestellt ist hier 8.

Beispiele:

```
COLLECT

COLLECT D1 ON U10
```

COLLISION–Anweisung

Format: **COLLISION *Typ*[,*Zeilennummer*]**

Zweck: Inaktiviert oder aktiviert die Programmunterbrechung für Sprite–Kollisionen und definiert im letzteren Fall eine Programmzeile für Programmverzweigung bei Kollisionen.

Typ Ganzzahliger Wert zwischen 1 und 3, der das Ereignis definiert, das zur Unterbrechung führen soll. Es gilt:

1 Sprite–Sprite–Kollision
2 Sprite–Anzeigedaten–Kollision
3 Lichtstift–Aktivierung

Zeilennummer Eine gültige Programmzeilennummer, zu der bei Unterbrechung verzweigt werden soll.

Bemerkungen: Wenn das angegebene Ereignis eintritt, führt der Interpreter die augenblicklich interpretierte Anweisung noch zuende und verzweigt dann mit GOSUB zu der angegebene Zeilennummer. Das dort zu durchlaufende Unterprogramm muß mit einer RETURN-Anweisung abgeschlossen sein. Anschließend wird das Programm mit der auf die unterbrochene Anweisung folgenden Anweisung fortgesetzt.

Wird ***Zeilennr*** weggelassen, wird die Programmunterbrechung für das spezifizierte Ereignis inaktiviert.

Es können mehrere verschiedene Unterbrechungsereignisse gleichzeitig aktiviert sein; es kann jedoch zu einer Zeit immer nur eine Unterbrechung bearbeitet werden. Unterbrechungsschachtelung ist also nicht möglich.

Es muß berücksichtigt werden, daß die Ursache für eine Programmunterbrechung eine Zeit lang wirksam sein kann, obwohl bereits eine andere Situation vorliegt oder die Unterbrechung inzwischen inaktiviert wurde.

Verschwindet ein Sprite vom sichtbaren Bereich des Bildschirms, kann es auch keine Unterbrechung mehr auslösen.

Um das Sprite zu bestimmen, das seit der letzten Abfrage eine Kollision verursacht hat, kann die BUMP–Funktion (s. dort) verwendet werden.

Beispiel:

```
100 COLLISION 2,500
```

Das Programm verzweigt zu einem Unterprogramm bei Zeile 500, sobald eine Kollision zwischen einem Sprite und angezeigten Daten erfolgt.

COLOR–Anweisung

Format: **COLOR *Farbquelle,Farbcode***

Zweck: Definiert für jede mögliche Farbquelle eine der 16 möglichen Farben.

Farbquelle Folgende Farbquellcodes sind verfügbar:

0	Hintergrund (40–Zeichen–Anzeige)
1	grafischer Vordergrund
2	grafischer Mehrfarbenmodus 1
3	grafischer Mehrfarbenmodus 2
4	Rand
5	Textfarbe
6	Hintergrund (80–Zeichen–Anzeige)

Farbcode Folgende Farbcodes sind verfügbar:

1	schwarz	9	hellbraun
2	weiß	10	braun
3	rot	11	rosa
4	grün	12	dunkelgrau
5	violett	13	grau
6	dunkelgrün	14	hellgrün
7	blau	15	hellblau
8	gelb	16	hellgrau

Bemerkungen: Die Farbwiedergabe kann bei verschiedenen Farbbildschirmen unterschiedlich ausfallen. Auf Schwarz/Weiß-Bildschirmen erscheinen die Farben als unterschiedliche Grautöne.

Beispiel:

```
COLOR 0,2:COLOR 1,3:COLOR 4,1
```

Der Hintergrund erscheint weiß, der Vordergrund (Text) rot und der Rahmen schwarz.

Color 0,2 ⇒ schwarz

CONCAT–Befehl

Format: **CONCAT [D*Quellaufw*,]*Quelldatei* TO [D*Ziellaufw*,]*Zieldatei* [ON U*Geräteadr*]**

Zweck: Fügt eine angegebene sequentielle Datei an das Ende einer anderen angegebenen sequentiellen Datei auf der Diskette im angegebenen Laufwerk an (s.a. Kapitel 6.9.3).

Quellaufw,Ziellaufw Ganzzahlige Werte von 0 oder 1. Voreingestellt ist hier 0.

Quelldatei,Zieldatei Zeichenkettenkonstanten in Anführungszeichen (") oder –variablen eine Länge von bis zu 16 Zeichen, die die anzufügende und die zu verlängernde sequentielle Datei bezeichnen.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15, der die Geräteadresse der Floppy–Disk angibt. Voreingestellt ist hier der Wert 8.

Beispiele:

```
CONCAT "DATEN1" TO "DATEN2"
```

Die sequentielle Datei DATEN1 wird an die sequentielle Datei DATEN2 auf Laufwerk 0 der Floppy–Disk mit der Geräteadresse 8 angefügt.

```
CONCAT D(QL),(D$1) TO D(ZL),(D2$)
```

Bei diesem Beispiel werden für Laufwerk– und Dateiangaben Variablen verwendet.

COPY–Befehl

Format: **COPY [D*Quellaufw*,]*Quelldatei* TO [D*Ziellaufw*,]*Zieldatei* [ON U*Geräteadr*]**

Zweck: Kopiert eine angegebene Quelldatei unter demselben oder einem neuen Namen auf die Diskette im angegebenen Laufwerk derselben Floppy–Disk–Einheit.

Quellaufw,Ziellaufw Ganzzahlige Werte von 0 oder 1. Voreingestellt ist hier 0.

Quelldatei,Zieldatei Zeichenkettenkonstanten in Anführungszeichen (") oder –variablen einer Länge von bis zu 16 Zeichen, die die zu kopierende und die kopierte Datei bezeichnen.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15, der die Geräteadresse der Floppy–Disk angibt. Voreingestellt ist hier der Wert 8.

Bemerkungen: Bei der Angabe der Quelldatei können die Jokerzeichen * und ? verwendet werden, um mehrere Dateien in einem Durchgang zu kopieren.

Beispiele:

```
COPY "DATEN" TO "ZIEL"
```

Die Datei DATEN auf Diskette in Laufwerk 0 der Floppy-Disk mit der Geräteadresse 8 wird unter dem Namen "ZIEL" dupliziert.

```
COPY "QUELLE" TO D1,"*"
```

Die Datei "QUELLE" wird von Laufwerk 0 auf Laufwerk 1 unter demselben Namen kopiert.

```
COPY D1,"ADR*" TO "*" ON U9
```

Auf der Floppy–Disk mit der Geräteadresse 9 werden alle Dateien auf Laufwerk 1, deren Namen mit "ADR" beginnen, unter Beibehaltung ihrer Namen auf Laufwerk 0 kopiert.

DCLEAR–Anweisung

Format: **DCLEAR D*Laufwerk* [ON U*Geräteadr*]**

Zweck: Alle für das angegebene Laufwerk offenen Floppy-Disk-Kanäle (**nicht** Dateien) werden geschlossen.

Laufwerk Ganzzahliger Wert von 0 oder 1.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15, der die Geräteadresse der Floppy–Disk angibt. Voreingestellt ist hier der Wert 8.

Bemerkungen: Wird ein Kanal z.B. mit der CMD–Anweisung (s. dort in Kapitel 5.2) eröffnet, so bleibt dieser Kanal aktiv, bis eine Ausgabe mit PRINT# erfolgt, oder eine DCLEAR–Anweisung gegeben wird.

Achtung
DCLEAR schließt **keine** geöffnete Datei. Das muß **vor** der Verwendung der DCLEAR–Anweisung erfolgen, falls Daten mit der CMD-Anweisung ausgegeben wurden.

Beispiel:

```
DCLEAR
```

Alle offenen Kanäle für Laufwerk 0 auf Gerät 8 werden geschlossen.

DCLOSE–Anweisung

Format: **DCLOSE** [#***log Dateinr***] [**ON U*Geräteadr***]

Zweck: Schließt die spezifizierte oder alle offenen Dateien auf der angeschlossenen Floppy–Disk (s.a. Kapitel 6.6).

log Dateinr Ein ganzzahliger Wert zwischen 1 und 255, unter dem die Datei mit der OPEN– (s. dort in Kapitel 5.2) oder DOPEN– (s. dort in diesem Kapitel) Anweisung auf Floppy–Disk eröffnet wurde.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15, der die Geräteadresse der Floppy–Disk angibt. Voreingestellt ist hier der Wert 8.

Beispiele:

```
DCLOSE
```

Alle für die Floppy–Disk mit der Geräteadresse 8 eröffneten Dateien werden geschlossen.

```
DCLOSE #5 ON U9
```

Die Datei mit der logischen Dateinummer 5 auf der Floppy-Disk mit der Geräteadresse 9 wird geschlossen.

DEC–Funktion

Format: ***v$*** **= DEC(*hex Zeichenkette*)**

Zweck: Liefert den Dezimalwert eines Zahlenwertes in hexadezimaler Schreibweise.

hex Zeichenkette Eine Zeichenkette in Anführungszeichen ("), die einen hexadezimalen Wert im Bereich zwischen 0 und FFFF darstellt.

Beispiel:

```
PRINT DEC("FFFF")
65535
READY.
```

DELETE–Befehl

Format: **DELETE** [*Zeilennr1*][*-Zeilennr2*]

Zweck: Der angegebene Zeilennummernbereich wird in dem im Hauptspeicher befindlichen Programm gelöscht.

Zeilennr1 Nummer der ersten zu löschenden Zeile.

Zeilennr2 Nummer der letzten zu löschenden Zeile.

Bemerkungen: Der DELETE–Befehl kann nur im Direktmodus verwendet werden.

Beispiele:

```
DELETE 10
```

Zeile 10 wird gelöscht.

```
DELETE 10-200
```

Zeilen 10 bis 200 werden gelöscht.

```
DELETE -200
```

Alle Zeilen vom Programmanfang bis Zeile 200 einschließlich werden gelöscht.

```
DELETE 200-
```

Alle Zeilen von einschließlich Zeile 200 bis um Programmende werden gelöscht.

DIRECTORY–Befehl

Format: **DIRECTORY** [D*Laufwerk*][,U*Geräteadr*]
[,*Dateiname*]

Zweck: Zeigt das Verzeichnis der Diskette auf dem spezifizierten Laufwerk oder Teile davon auf dem Bildschirm an (s.a. Kapitel 6.5).

Laufwerk Ganzzahliger Wert von 0 oder 1. Voreingestellt ist hier der Wert 0:

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15, der die Geräteadresse der Floppy–Disk angibt. Voreingestellt ist hier der Wert 8.

Dateiname Wahlweise ein Dateiname als Zeichenkettenkonstante in Anführungsstrichen ("), für den der Verzeichniseintrag angezeigt werden soll. Um Dateien mit gleichen Namensteilen anzuzeigen, dürfen die Jokerzeichen * und ? (s.a. Kapitel 6.3) verwendet werden.

Bemerkungen: Die Anzeige langer Verzeichnisse führt zum Aufrollen des Bildschirms. Mit Hilfe der NO SCROLL–Taste (s. Kapitel 5.2) kann die Anzeige angehalten werden, mit der C= (Commodore)–Taste kann sie verlangsamt werden.

Beispiele:

```
DIRECTORY
```

Alle Verzeichniseinträge von der Diskette in Laufwerk 0 der Floppy–Disk mit der Geräteadresse 8 werden angezeigt.

```
DIRECTORY D1,U9,"DATEN"
```

Der Verzeichniseintrag der Datei "DATEN" auf der Diskette in Laufwerk 1 der Floppy–Disk mit der Geräteadresse 9 wird, falls vorhanden, angezeigt.

```
DIRECTORY "AB*"
```

Die Verzeichniseinträge für alle Dateien, deren Namen mit AB beginnen, auf der Diskette in Laufwerk 0 der Floppy-Disk mit der Geräteadresse 8 werden angezeigt.

```
DIRECTORY "DATEI???.ARB"
```

Die Verzeichniseinträge für alle Dateien, deren Namen zwischen den Zeichen DATEI und .ARB drei beliebige Zeichen enthalten, auf der Diskette in Laufwerk 0 der Floppy–Disk mit der Geräteadresse 8 werden angezeigt.

Achtung:
Um ein Diskettenverzeichnis z. B. von Laufwerk 0 auf der Floppy–Disk mit der Geräteadresse 8 auf dem Drucker mit der Geräteadresse 4 auszugeben, muß folgende Anweisungssequenz im Direktmodus eingegeben werden:

```
LOAD"$0",8
OPEN4,4:CMD4:LIST
PRINT#4:CLOSE4
```

DLOAD–Befehl

Format: **DLOAD *Dateiname*[,D*Laufwerk*]**
[,U*Geräteadr*]

Zweck: Lädt ein BASIC–Programm von einer angegebenen Diskette in den Hauptspeicher des Rechners s.a. Kapitel 6.6).

Dateiname Eine Zeichenkettenkonstante in Anführungsstrichen (") oder eine –Variable in Klammern () einer Länge von bis zu 16 Zeichen, die den Programmdateinamen bezeichnet.

Laufwerk Ein ganzzahliger Wert von 0 oder 1. Voreingestellt ist hier 0.

Geräteadr Ein ganzzahliger Wert zwischen 4 oder 15, der die Geräteadresse der angeschlossenen Floppy-Disk bezeichnet. Voreingestellt ist hier der Wert 8.

Bemerkungen: Innerhalb eines BASIC–Programmes dient der DLOAD-Befehl zum Aufruf und Start eines anderen BASIC–Programmes, das das aufrufende Programm überlagert. Alle Variablen des aufrufenden Programms bleiben erhalten.

Wird das spezifizierte Programm nicht gefunden, wird die Fehlermeldung **FILE NOT FOUND ERROR** angezeigt.

DO–Anweisung

Formate:

DO [UNTIL *logischer Ausdruck*]
.
. Anweisungen
.
. . . **[EXIT]**
LOOP [UNTIL *logischer Ausdruck*]

DO [WHILE *logischer Ausdruck*]
.
. Anweisungen
.
. . . **[EXIT]**
LOOP [WHILE *logischer Ausdruck*]

Zweck: Definiert und steuert den Ablauf von Programmschleifen.

Bemerkungen: Wenn kein Zusatz UNTIL oder WHILE in Verbindung mit logischen Ausdrücken angegeben ist, werden alle Anweisungen der Schleife unendlich oft ausgeführt.

Wird der Zusatz EXIT in die Schleifenanweisungen eingefügt, so wird die Schleife an dieser Stelle verlassen, und das Programm wird mit der auf LOOP folgenden Anweisung fortgesetzt.

DO . . . LOOP–Programmschleifen können nach denselben Regeln geschachtelt werden, wie FOR . . . NEXT-Schleifen (s. dort in Kapitel 5.2).

Wird der Zusatz UNTIL verwendet, so wird die Schleife solange ausgeführt, bis der nach UNTIL angegebene logische Ausdruck wahr (–1) wird.

Wird der Zusatz WHILE verwendet, so wird die Schleife solange ausgeführt, bis der nach WHILE angegebene logische Ausdruck falsch (0) wird.

Beispiel 1:

```
100 X=10
110 DO UNTIL X=0 OR X=1
120 PRINT "X IST NOCH NICHT 0 ODER 1"
130 X=X-1
140 LOOP
150 . . .
```

Die Schleife wird neunmal durchlaufen, dann ist X=1 wahr und das Programm wird mit Zeile 150 fortgesetzt.

Beispiel 2:

```
100 DO WHILE A$="":GET A$:LOOP
```

Die Programmausführung wird solange angehalten, bis eine Taste gedrückt wird.

DOPEN–Anweisung

Format: **DOPEN #*log Filenr*, *Dateiname*[,L*Länge*] [,D*Laufw*][,U*Geräteadr*][,W]**

Zweck: Eröffnet eine sequentielle oder Relativ–Datei auf Diskette für Daten–Ein– oder –Ausgabe (s.a. Kapitel 6.6).

log Filenr Ein ganzzahliger Wert zwischen 1 und 255, der der eröffneten Datei als Kennummer zugeordnet wird. Bei Werten kleiner als 128 wird nach jeder PRINT#–Anweisung ein Wagenrücklaufcode ausgegeben. Bei Werten über 127 wird zusätzlich ein Zeilenvorschubcode ausgegeben.

Dateiname Eine Zeichenkettenkonstante in Anführungsstrichen (") oder –variable einer Länge von bis zu 16 Zeichen, die den Namen der zu eröffnenden Datei bezeichnet.

Länge Ein ganzzahliger Wert zwischen 1 und 254 für die Länge der logischen Sätze bei Relativ–Dateien.

Laufw Ein ganzzahliger Wert von 0 oder 1. Voreingestellt ist hier 0.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15 für die Geräteadresse der angeschlossenen Floppy-Disk. Voreingestellt ist hier 8.

W Wird dieser Parameter angegeben, so wird eine sequentielle Datei zum Schreiben eröffnet, andernfalls zum Lesen.

Bemerkungen: Werden die Parameter als Variablen übergeben, so müssen diese in Klammern gesetzt werden.

L*Länge* und **W** dürfen nicht zusammen in einer DOPEN-Anweisung angegeben werden.

Werden zwei Dateien unter derselben logischen Dateinummer eröffnet, wird die Fehlermeldung FILE ALREADY OPEN angezeigt.

Beispiele:

```
DOPEN #2,"DATEN"
```

Die sequentielle Datei DATEN auf Laufwerk 0 der Floppy-Disk mit der Geräteadresse 8 wird zum Lesen eröffnet.

```
DOPEN #3,"ADRESSEN",D1,U9,W
```

Die sequentielle Datei ADRESSEN auf Laufwerk 1 der Floppy–Disk mit der Geräteadresse 9 wird zum Schreiben eröffnet.

```
DOPEN #5,"KLIENTEN",L112
```

Die Relativ–Datei KLIENTEN mit einer logischen Satzlänge von 112 Bytes wird auf Laufwerk 0 der Floppy-Disk mit der Geräteadresse 8 eröffnet.

DRAW–Anweisung

Format: **DRAW** [*Farbquel*],[*x1,y1*] **TO** *x2,y2*...

Zweck: Verbindet beliebig viele spezifizierte Koordinaten durch eine Linie.

Farbquelle Ein ganzzahliger Wert zwischen 0 und 3, der die Farbe definiert, in der die Linie(n) gezeichnet werden soll(en). Voreingestellt ist die Vordergrundfarbe (1). Es bedeuten:

0 Hintergrundfarbe
1 Vordergrund– (Zeichen–)Farbe
2 Zusatzfarbe 1
3 Zusatzfarbe 2

x1,y1 Koordinaten des Startpunktes der Linie.

x2,y2 Koordinaten des Endpunktes der Linie.

Bemerkungen: Es können soviele Koordinaten mit davor gestelltem Parameter TO spezifiziert werden, wie in einer Programmzeile Platz haben.

Der Endpunkt der vorhergehenden Linie ist dann gleichzeitig der Startpunkt für die folgende Linie.

Sind Start– und Endkoordinaten gleich, wird ein Punkt gezeichnet.

Wird die Startkoordinate nicht angegeben, so gilt die aktuelle Position des grafischen Cursors (Pixel–Cursor) als Startpunkt.

Beispiele:

```
DRAW 1,100,50
```

Zeichnet einen Punkt, da keine Endkoordinate angegeben ist.

```
100 DRAW ,10,10 TO 100,60
110 DRAW TO 25,30
```

Zeichnet einen Winkel.

```
DRAW ,10,10 TO 10,60 TO 100,60 TO 10,10
```

Zeichnet eine Figur.

DSAVE–Befehl

Format: **DSAVE** ***Dateiname***[**,D*Laufwerk***]
[**,U*Geräteadr***]

Zweck: Speichert ein BASIC–Programm aus dem Hauptspeicher auf Diskette im angegebenen Laufwerk s.a. Kapitel 6.2).

Dateiname Eine Zeichenkettenkonstante in Anführungsstrichen (") oder eine –variable in Klammern () mit einer Länge von bis zu 16 Zeichen, die den Programmdateinamen bezeichnet.

Laufwerk Ein ganzzahliger Wert von 0 oder 1. Voreingestellt ist hier 0.

Geräteadr Ein ganzzahliger Wert zwische n 4 oder 15, der die Geräteadresse der angeschlossenen Floppy-Disk bezeichnet. Voreingestellt ist hier der Wert 8.

Bemerkungen: Werden die Parameter als Variablen übergeben, so müssen diese in Klammern gesetzt werden.

Nach Ausführung von DSAVE kehrt der Interpreter auf jeden Fall in den Direktmodus zurück.

Zur Speicherung von Programmen auf Kassette muß die SAVE–Anweisung (s. dort in Kapitel 5.2) verwendet werden.

Existiert die angegebene Datei bereits auf Diskette, so wird die Fehlermeldung FILE EXISTS erzeugt, die mit PRINT DS$ angezeigt werden kann.

Beispiele:

```
DSAVE"PROG1"
```

Das im Hauptspeicher befindliche Programm wird unter dem Namen PROG1 auf die Diskette in Laufwerk 0 von Gerät 8 gespeichert.

```
DSAVE(A$),D(LW),U(GA)
```

Hier werden die Parameter als Variablen übergeben.

DS– und DS$–Systemvariablen

Format: ***v*= DS**
***v$*= DS$**

Zweck: Liefert den Fehlerstatus als Code (DS) und als komplette Statusmeldung (DS$) des zuletzt angesprochenen Floppy–Disk–Gerätes.

Bemerkungen: Ein Statuscode von 0 bedeutet kein Fehler.

Die Statusmeldung ist eine Zeichenkette und besteht aus 4 Teilen:

Statuscode
Statustext
Spur–Nr., bei der der Fehler auftrat
Sektor–Nr., bei der der Fehler auftrat

Durch eine Fehlerabfrage wird der Fehlerstatus zurückgesetzt.

Da DS und DS$ Systemvariablen sind, dürfen sie nicht als anwenderspezifische Variablen in einem Programm verwendet werden. Andernfalls würde der Fehler SYNTAX ERROR angezeigt.

Beispiel:

```
PRINT DS$:PRINT DS
32, SYNTAX ERROR,00,00
32
READY.
```

Das zuletzt angesprochene Floppy–Disk–Gerät zeigt einen Syntax–Fehler, weil die übermittelte Befehlszeichenkette zu lang war.

DVERIFY–Befehl

Format: **DVERIFY** ***Dateiname***[**,D*Laufwerk***]
[**,U*Geräteadr***]

Zweck: Vergleicht ein Programm im Hauptspeicher byteweise mit einer Programmdatei auf einer angegebenen Diskette.

Dateiname Eine Zeichenkettenkonstante in Anführungsstrichen (") oder eine –Variable in Klammern () einer Länge von bis zu 16 Zeichen, die den Programmdateinamen bezeichnet.

Laufwerk Ein ganzzahliger Wert von 0 oder 1. Voreingestellt ist hier 0.

Geräteadr Ein ganzzahliger Wert zwischen 4 oder 15, der die Geräteadresse der angeschlossenen Floppy-Disk bezeichnet. Voreingestellt ist hier der Wert 8.

Bemerkungen: Wenn nach Ausführung eines SAVE– oder DSAVE-Befehls ein grafischer Bereich reserviert oder wieder freigegeben wird, liefern die Befehle VERIFY oder DVERIFY die Fehlermeldung VERIFY ERROR, obwohl das Programm in Ordnung ist. Dies entsteht dadurch, daß durch Reservierung oder Freigabe eines grafischen Bereiches ein BASIC–Programm aus seiner ursprünglichen Speicherposition verschoben wird, wodurch die Adreßzeiger innerhalb der BASIC-Zeilen verändert werden und

eine byteweise Übereinstimmung mit dem gespeicherten Programm nicht mehr gegeben ist.

Beispiele:

```
DVERIFY"PROG1"
```

Das Programm PROG1 auf Laufwerk 0 in der Floppy-Disk 8 wird mit dem hauptspeicherresidenten Programm byteweise verglichen.

```
DVERIFY"MULT",D1,1
```

Das Programm MULT auf Laufwerk 1 der Floppy–Disk 8 wird mit einem hauptspeicherresidenten Maschinenspracheprogramm verglichen, das nicht am Anfang des BASIC–Programmspeichers beginnt.

EL–Systemvariable

Format: *v*= EL

Zweck: Liefert die Nummer der Programmzeile, in der ein Fehler aufgetreten ist.

Bemerkungen: Da EL eine Systemvariablen ist, darf sie nicht als anwenderspezifische Variable in einem Programm verwendet werden. Andernfalls würde der Fehler SYNTAX ERROR angezeigt.

Beispiele:

```
800 IF EL=100 THEN 1000
```

Wenn EL in einer IF–Abfrage verwendet wird, muß die Vergleichsnummer rechts von der EL–Variablen stehen, damit sie bei einer ggfs. durchgeführten Renumerierung berücksichtigt werden kann.

ENVELOPE–Anweisung

Format: **ENVELOPE** ***n*** [,[***An***][,[***Ab***][,[***Ha***]
[,[***Au***][,[***Wf***][,***Ib***]]]]]]

Zweck: Definiert eine von 10 möglichen Hüllkurven für Musikinstrumente.

n Nummer der Hüllkurve (0 bis 9).
An Anschlagzeit (0 bis 15).
Ab Abschwellzeit (0 bis 15).
Ha Haltezeit (0 bis 15).
Au Ausklingzeit (0 bis 15).
Wf Wellenform:
- 0 Dreieck
- 1 Sägezahn
- 2 Rechteck
- 3 Rauschen
- 4 Ringmodulation

Ib Impulsbreite (0 bis 4095).

Bemerkungen: Ein nicht spezifizierter Parameter behält seinen momentanen Wert.

Die Impulsbreite ist nur im Zusammenhang mit der Rechteck–Wellenform (***Wf***=2) definierbar und wird durch die Formel
PULSBREITE=***Ib***/40.95 % bestimmt, so daß ***Ib***=2048 eine Rechteckwelle erzeugt.

Nach dem Einschalten des Rechners werden für die Parameter folgende Werte voreingestellt:

n	*An*	*Ab*	*Ha*	*Au*	*Wf*	*Ib*	**Instrument**
0	0	9	0	0	2	1536	Klavier
1	12	0	12	0	1		Akkordeon
2	0	0	15	0	0		Zirkusorgel
3	0	5	5	0	3		Trommel
4	9	4	4	0	0		Flöte
5	0	9	2	1	1		Gitarre
6	0	9	0	0	2	512	Cembalo
7	0	9	9	0	2	2048	Orgel
8	8	9	4	1	2	512	Trompete
9	0	9	0	0	0		Xylophon

Weitere Einzelheiten zur ENVELOPE–Anweisung entnehmen Sie bitte dem Abschnitt „Musikerzeugung mit dem C128“ ab Seite 4–156 ff.

ER–Systemvariable

Format: ***v*= ER**

Zweck: Liefert den Code des zuletzt vom Interpreter diagnostizierten Fehlers.

Im Kapitel 8.1 sind die Fehlermeldungen zusammen mit den ihnen zugeordneten Fehlercodes aufgeführt.

Bemerkungen: Da ER eine Systemvariablen ist, darf sie nicht als anwenderspezifische Variable in einem Programm verwendet werden. Andernfalls würde der Fehler SYNTAX ERROR angezeigt.

Beispiele:

```
100 IF ER=30 THEN 1000
```

Wenn die STOP–Taste gedrückt wird, verzweigt das Programm nach Zeile 1000.

ERR$–Funktion

Format: *v$*=ERR$(*n*)

Zweck: Liefert den Text des durch *n* spezifizierten Fehlers.

n Ein ganzzahliger Wert zwischen 1 und 127, der den Fehlercode angibt (s. Kapitel 8.1).

Beispiel:

```
10 TRAP 1000
.
. BASIC–Anweisungen
.
1000 REM ANALYSE DER FEHLERMELDUNGEN
1001 REM BEI SYNTAX-ERROR PROGRAMM-
     STOP
1010 IF ER=11 THEN PRINT EL,ERR$:STOP
1011 REM WENN KEIN SYNTAX ERROR, WIRD
1012 REM MELDUNG GEPRUEFT UND GEDRUCKT
1020 IF ER=9 THEN PRINT EL,ERR$(ER):
     RESUME 100
1030 IF ER=30 THEN RESUME 150
1040 RESUME 975
```

Die übrigen zur Fehlerverarbeitung benötigten Anweisungen sind bei RESUME und TRAP beschrieben.

FAST-Befehl

Format: **FAST**

Zweck: Schaltet den Prozessor vom Betrieb im 1–Megahertz-Takt auf Betrieb im 2–MegahertzTakt um. Der Rechner wird damit doppelt so schnell.

Bemerkungen: Der 40–Zeichen–Bildschirm wird durch den FAST-Befehl inaktiviert, d.h. Text– oder Grafikausgaben sind nur noch auf dem 80–Zeichen–Bildschirm sichtbar.

Sollten Sie ausschließlich einen Fernsehempfänger oder einen Monitor mit 40–Zeichen–Darstellung benutzen, schalten Sie bitte vor der Ausgabe von Texten oder Grafik mit dem Befehl SLOW (s. dort) in den 1–MHz–Betrieb zurück.

FETCH–Befehl

Format: **FETCH *Bytes,Intadr,Extadr,Bank***

Zweck: Überträgt eine wählbare Anzahl von Bytes aus einer Speichererweiterungs–Bank in den BASIC–Arbeitsspeicher.

Bytes Ein ganzzahliger Wert zwischen 0 und 65535, der die Anzahl der zu übertragenden Bytes angibt.

Intadr Ein ganzzahliger Wert zwischen 0 und 65535, der die Adresse im BASIC–Arbeitsspeicher angibt, ab dder die Bytes abgelegt werden sollen.

Extadr Ein ganzzahliger Wert zwischen 0 und 65535, der die Adresse in der Speicherbank angibt, ab der die Bytes geholt werden sollen.

Bank Ein ganzzahliger Wert zwischen 0 und 7, der die Speicherbank angibt, aus der die Bytes geholt werden sollen.

Bemerkungen: Dieser Befehl kann nur ausgeführt werden, wenn eine Ram–Disk an den Rechner angeschlossen ist. Die Ram Disk decodiert den entsprechenden Befehl, und führt ihn aus.

Der Parameter Intadr bezieht sich auf die Bank, die mit dem BANK Befehl eingestellt wurde. Voreingestellt ist Bank 15.

Sollen z. B. Daten aus Bank 0 in der Ram Disk abgespeichert werden, so muß vor dem STASH Befehl 'BANK 0' eingegeben werden.

Beispiel:

```
FETCH 1000,52000,2000,7
```

Überträgt 1000 Bytes aus Speicherbank 7 ab Adresse 2000 in den BASIC–Arbeitsspeicher ab Adresse 52000.

Achtung:
Um diesen Befehl ausführen zu können, brauchen Sie das Steckmodul Ram–Disk.

FILTER–Anweisung

Format: **FILTER** [***Freq***][,[***Tp***][,[***Bp***][,[***Hp***]
[,[***Res***]]]]]

Zweck: Definiert die Parameter für Klangfilter.

Freq Ein ganzzahliger Wert zwischen 0 und 2047, der die Filtergrenzfrequenz festlegt.

Tp,Bp,Hp Ein ganzzahliger Wert von 0 oder 1, der das Tiefpaß–, Bandpaß– oder Hochpaß–Filter ein– (1) oder ausschaltet (0).

Res Ein ganzzahliger Wert zwischen 0 und 15, der die Resonanz festlegt.

Bemerkungen: Bei nicht angegebenen Parametern bleibt der aktuelle Wert gültig.

Die tatsächliche Filtergrenzfrequenz errechnet sich aus
GRENZFREQ = ***Freq****5.8 + 30.
Dieser Wert ist jedoch nur ein Anhaltswert und kann sich von Rechner zu Rechner geringfügig ändern.

Die Filterausgänge sind additiv. So können durch gemeinsame Auswahl von Tief– und Hochpaßfiltern Kerbfilter– oder Bandsperr–Effekte erzielt werden.

Um hörbare Filtereffekte zu erzielen, muß mindestens ein Filter gewählt und mindestens eine Tonfrequenz durch das Filter geführt werden.

Weitere Einzelheiten zur FILTER–Anweisung entnehmen Sie bitte dem Abschnitt „Musikerzeugung mit dem C128“ ab Seite 4–162.

FRE–Funktion

Format: *v*= **FRE(*Bank*)**

Bedeutung: Liefert die Anzahl der noch nicht benutzten Speicherplätze im BASIC–Programm– oder –Arbeitsspeicher.

Bank Ein ganzzahliger Wert von 0 oder 1, der die Speicherbank bezeichnet, deren ungenutzte Speicherplätze abgefragt werden sollen.

Bemerkungen: Wird FRE(1) aufgerufen, so wird der Speicherbereich für Zeichenketten in der Speicherbank 1 organisatorisch bereinigt, ehe die Anzahl der freien Speicherplätze übergeben wird. Dieser Speicherbereich wird nämlich vom Interpreter dynamisch verwaltet, d.h. Mehrfachzuweisungen von Zeichenketten zu ein und derselben Variablen führen zu immer neuem Anlegen dieser Ketten im Speicher.

GETKEY–Anweisung

Format: **GETKEY** ***Liste von Zeichenkettenvariablen***

Zweck: Die GETKEY–Anweisung ähnelt der GET–Anweisung (s. dort in Kapitel 5.2). Sie wartet jedoch solange, bis eine Taste auf der Tastatur gedrückt wird, und weist dann den ASCII–Code der gedrückten Taste als 1–Byte–Zeichenkette der spezifizierten Variablen zu. Werden mehrere Zeichenkettenvariablen, getrennt durch Komma angegeben, so müssen entsprechend viele Tasten betätigt werden.

Liste von Zeichenkettenvariablen Eine oder mehrere, durch Komma getrennte, Zeichenkettenvariablen.

Bemerkungen: Die GETKEY–Anweisung kann nur im Programm–Modus verwendet werden. Andernfalls wird die Fehlermeldung ILLEGAL DIRECT angezeigt.

Beispiele:

```
100 GETKEY A$,B$,C$
```

Die ASCII–Codes von drei gedrückten Tasten werden jeweils als 1–Byte–Zeichenkette den Variablen A$, B$ und C$ zugewiesen.

```
100 GETKEY A$
110 IF A$<"A" OR A$>"B" THEN 100
120 ON ASC(A$)-64 GOTO 1000,2000
```

Das Programm verzweigt nur nach Zeile 1000 oder 2000, wenn entweder die Taste A oder B gedrückt wird.

GO64–Befehl

Format: **GO64**

Zweck: Der Rechner wird vom C128–Modus in den C64–Modus umgeschaltet.

Bemerkungen: Der Interpreter fordert durch die Anzeige

```
ARE YOU SURE?
```

vom Anwender eine Bestätigung für den Befehl. Wird Y (Yes) eingegeben, so wird das momentan im Speicher befindliche Programm gelöscht und der Rechner wird in den C64–Modus umgeschaltet. Bei N (No) bleibt der C128–Modus aktiv und es wird am bestehenden Programm nichts verändert.

Dieser Befehl kann sowohl im Direkt– als auch im Programm–Modus gegeben werden.

GRAPHIC–Anweisung

Formate: **GRAPHIC** ***Modus***[,[***Lösch***],***Textz***]
GRAPHIC CLR

Zweck: Aktiviert Grafikmodus und reserviert Grafikspeicher (bit mapped) bzw. gibt Grafikspeicher wieder frei.

Modus Ein ganzzahliger Wert zwischen 0 und 5. Folgende Modi sind wählbar:

0 Text mit 40 Zeichen/Zeile.
1 hochauflösende Grafik (320x200 Punkte).
2 hochauflösende Grafik, geteilter Bildschirm für Grafik und Text.
3 Mehrfarbengrafik (160x200 Punkte)
4 Mehrfarbengrafik, geteilter Bildschirm für Grafik und Text.
5 Text mit 80 Zeichen/Zeile.

Lösch Ein ganzzahliger Wert von 0 oder 1, der bestimmt, ob der Bildschirm beim Aufruf der Grafikmodi 1 bis 4 gelöscht werden soll (1) oder nicht (0).

Textz Ein ganzzahliger Wert zwischen 0 und 24, der bei den Modi 2 und 4 die Zeile festlegt, bei der der Text–Teilbildschirm beginnt. Voreingestellt ist hier 19.

GRAPHIC CLR gibt den durch die GRAPHIC *Modus*-Anweisung reservierten Speicher wieder frei.

Bemerkungen: Im Kapitel 4.7 werden hochauflösende und Mehrfarbengrafik ausführlich beschrieben.

Bei den Modi 1 bis 4 wird ein 9–KByte–Speicherbereich am Anfang des BASIC–Programmspeichers als Grafikspeicher reserviert. Der Beginn des BASIC-Programmspeichers wird dadurch nach oben verschoben.

Dieser Bereich bleibt auch reserviert, wenn wieder der Text–Modus (1 oder 5) eingeschaltet wird und wird erst durch die GRAPHIC CLR–Anweisung wieder freigegeben.

Beispiele:

```
GRAPHIC 1
```

Schaltet auf hochauflösende Grafik um.

```
GRAPHIC 4,1,15
```

Schaltet auf Mehrfarbengrafik mit geteiltem Bildschirm und setzt den Anfang des Textschirms auf Zeile 16 (Zeilen beginnen mit 0).

```
GRAPHIC CLR
```

Der reservierte Grafikspeicher wird wieder freigegeben.

GSHAPE–Anweisung

Format: **GSHAPE** ***Zeikett***[,[*x,y*][,***Modus***]]]

Zweck: Überträgt den Wert einer angegebenen Zeichenkette als binäre Bildinformation auf den Grafik–Bildschirm (s.a. MOVSPR–Anweisung).

Zeikett Eine Zeichenkettenvariable oder – konstante, die die abzubildende Bildinformation binär enthält (s. SSHAPE–Anweisung).

x,y Die skalierten Bildschirmkoordinaten der linken oberen Ecke des darzustellenden Bildes.

Modus Ein ganzzahliger Wert zwischen 0 und 4, der die Art der Darstellung bestimmt. Folgende Modis sind verfügbar:

0 Die Daten aus der Zeichenkette werden genau so, wie sie mit SSHAPE gespeichert wurden, abgebildet.
1 Das Bild wird invertiert.
2 Dem auf dem Schirm vorhandenen Bild wird die Bildinformation aus der Zeichenkette überlagert (logisches ODER).
3 Es werden nur dann Bildpunkte übertragen, wenn unter den zu übertragenden Bildpunkten bereits Bildpunkte auf dem Schirm existieren (logisches UND).
4 Es werden im existierenden Bild dort die Bildpunkte invertiert, wo im zu übertragenden Bild Bildpunkte existieren. Dadurch kann ein Objekt über den Bildschirm bewegt werden, ohne den Hintergrund zu löschen.

HEADER–Befehl

Format: **HEADER *Name*,D*Laufwerk*[,I*Kennung*]**
[,U*Gerät*]

Zweck: Formatiert eine Diskette im angegebenen Laufwerk der gewählten Floppy–Disk neu oder löscht das Inhaltsverzeichnis (s.a. Kapitel 6.1).

Name Eine Zeichenkettenkonstante in Anführungszeichen (") oder eine –variable einer Länge von maximal 16 Zeichen, die den Namen der Diskette bezeichnet.

Laufwerk Ein ganzzahliger Wert von 0 oder 1 für die Laufwerksbezeichnung. 0 ist voreingestellt.

Kennung Zwei beliebige Zeichen für die Diskettenkennung. Diese Kennung wird bei der Formatierung in jeden Sektor eingetragen. Für jede Diskette sollte unbedingt eine eigene Kennung vergeben werden.

Achtung Wegen der besonderen Art und Weise, in der der Interpreter die Floppy–Disk–Befehle interpretiert, dürfen für den Parameter ***Kennung*** keine BASIC–Schlüsselworte wie ON, FN, TO usw. oder Abkürzungen von Schlüsselworten wie aB, cO usw. verwendet werden. Andernfalls wird die Fehlermeldung SYNTAX ERROR angezeigt.

Gerät Ein ganzzahliger Wert zwischen 4 und 15 für die Geräteadresse der angeschlossenen Floppy-Disk. 8 ist voreingestellt.

Bemerkungen: Da bei der Formatierung alle ggfs. vorher auf der Diskette gespeicherten Daten zerstört werden, wird der Anwender durch die Anfrage

```
ARE YOU SURE?
```

um Bestätigung gebeten. Die Antwort Y (Yes) führt den HEADER–Befehl aus, jede andere nicht.

Soll eine bereits formatierte Diskette nur gelöscht werden, wird beim HEADER–Befehl der Parameter ***IKennung*** weggelassen. Die Diskette erhält dann nur den neuen Namen, behält aber die alte Kennung.

Werden als Parameter Variablen verwendet, so müssen diese in Klammern gesetzt werden.

Beispiele:

```
HEADER"DATEN",D0,IDA
```

Die Diskette in Laufwerk 0 der Floppy–Disk mit der Geräteadresse 8 wird mit der Kennung DA neu formatiert und erhält den Namen DATEN.

```
HEADER"PROGRAMME",IP1,D1,U9
```

Die Diskette in Laufwerk 1 der Floppy–Disk mit der Geräteadresse 9 wird mit der Kennung P1 neu formatiert und erhält den Namen PROGRAMME.

```
HEADER(N$),I(K$),D(L),U(G)
```

Hier werden für alle Parameter Variablen verwendet.

```
HEADER"STAMMDATEN",D0
```

Die Diskette in Laufwerk 0 der Floppy–Disk mit der Geräteadresse 8 wird nur gelöscht, erhält den Namen STAMMDATEN, und behält ihre alte Kennung bei.

HELP–Befehl

Format: **HELP**

Zweck: Zeigt nach einer Fehlermeldung des Interpreters die fehlerhafte Programmzeile auf dem Bildschirm an und stellt den fehlerhaften Teil invers beim 40–Zeichen– und unterstrichen beim 80–Zeichen–Bildschirm dar (s.a. HELP–Taste in Kapitel 4.1).

Bemerkungen: Der HELP–Befehl (oder Drücken der HELP–Taste) hat bei Fehlern, die bei im Direktmodus eingegebenen Anweisungen diagnostiziert werden, keine Wirkung.

HEX$–Funktion

Format: ***v$*** **= HEX$(*n*)**

Zweck: Liefert eine Zeichenkette, die die hexadezimale Darstellung des Wertes des numerischen Ausdruckes ***n*** enthält.

n Beliebiger numerischer Ausdruck, dessen ganzzahliger Wert gewandelt wird und zwischen 0 und 65535 liegen muß.

Beispiel:

```
100 X=23:Y=1456
110 PRINT X;HEX$(X),Y;HEX$(Y)
RUN
23 17       1456 5B0
READY.
```

IF–Anweisung

Format: **IF *Ausdr* THEN *Anw* [:ELSE *Anw*]**

Zweck: Erlaubt die Verzweigung in verschiedene Programmteile, abhängig vom logischen Wahrheitsgehalt eines numerischen Ausdruckes. Ein numerischer Ausdruck ist logisch "wahr", wenn sein Wert von Null verschieden ist und logisch "falsch", wenn sein Wert 0 ist.

Ausdr Jeder beliebige numerische Ausdruck.

Anw Eine einzelne Anweisung, eine Folge von Anweisungen durch Doppelpunkte voneinander getrennt oder einfach eine Zeilennummer.

Bemerkungen: Ist der numerische Ausdruck logisch "wahr", so wird (werden) die Anweisung(en) hinter THEN ausgeführt. Ist er dagegen logisch "falsch", so wird (werden) die Anweisung(en) hinter THEN ignoriert und die Anweisung(en) hinter ELSE, falls vorhanden, oder die nächste Programmzeile ausgeführt.

Beim Test auf Gleichheit muß die Ungenauigkeit auf Grund der internen Gleitkommadarstellung berücksichtigt werden. In kritischen Fällen sollte der Test über einen Bereich erfolgen, innerhalb dessen Grenzen die Genauigkeit schwankt. Z.B. liefert
IF ABS(X–2.5) < 1.06E–6 THEN . . .
das Ergebnis "wahr", wenn X = 2.5 mit einem relativen Fehler von weniger als 1.06E–6 ist.

Die IF . . . THEN . . .:ELSE–Anweisung muß in einer Programmzeile stehen. Weder THEN noch ELSE dürfen in separaten Zeilen stehen, es sei denn, es wird eine BEGIN–Anweisung in die IF–Anweisung eingefügt (s. Beispiel 2 bei der BEGIN–Anweisung).

Ist die Anzahl von THEN– und von ELSE–Angaben unterschiedlich, so bezieht sich ELSE immer auf das letzte davorstehende THEN.

Beispiel 1:

```
100 NMSG$="DER WERT MUSS POSITIV SEIN"
110 PMSG$="DIE WURZEL IST"
120 INPUT "BITTE ZAHL EINGEBEN";N
130 IF N<0 THEN PRINT NMSG$:GOTO 120
    :ELSE PRINT PMSG$;SQR(N)
140 INPUT "NOCH EINE ZAHL (J/N)";Z$
150 IF Z$="J" THEN 120:ELSE END
```

Es wird die Wurzel einer eingegebenen Zahl nur dann ermittelt und gedruckt, wenn die Zahl positiv ist.

Beispiel 2:

```
100 IF X>Y THEN PRINT "GROESSER"
    :ELSE IF Y>X THEN PRINT "KLEINER"
    :ELSE PRINT "GLEICH"
```

Dies ist ein Beispiel für verschachtelte IF–Anweisungen.

Beispiel 3:

```
100 IF A=B THEN IF B=C THEN PRINT "A=C"
    :ELSE PRINT "A<>C"
```

Falls A ungleich B ist, wird der zweite Vergleich nicht mehr ausgeführt.

INSTR–Funktion

Format: *v*= **INSTR**(*x$*,*y$*[,*n*])

Zweck: Sucht und liefert die Position des ersten Auftretens der Teilzeichenkette ***y$*** in ***x$***.

x$ Beliebiger Zeichenkettenausdruck, dessen Wert die zu durchsuchende Zeichenkette darstellt.

y$ Beliebiger Zeichenkettenausdruck, dessen Wert die gesuchte Teilzeichenkette darstellt.

n Ein numerischer Ausdruck, dessen ganzzahliger Wert zwischen 1 und 255 liegen muß und der die Startposition in der zu durchsuchenden Zeichenkette angibt. Voreingestellt ist hier 1.

Bemerkungen: In den folgenden Fällen liefert INSTR den Wert 0:

n ist größer als LEN(***x$***)
x$ ist leer
y$*** ist nicht in ***x$ enthalten

Beispiel:

```
PRINT INSTR("COMMODORE","R")
8
READY.
```

JOY–Funktion

Format: ***v*=JOY(*n*)**

Zweck: Liefert einen Wert für die Position sowie den Zustand des Feuerknopfes eines wählbaren Joysticks (Spielpult).

n Ein ganzzahliger Wert von 1 oder 2, der den gewünschten Joystick bezeichnet.

Der von der JOY–Funktion übergebene Wert hat folgende Bedeutung:

0 Mittelstellung
1 nach vorn
2 nach diagonal rechts vorn
3 nach rechts
4 nach diagonal rechts hinten
5 nach hinten
6 nach diagonal links hinten
7 nach links
8 nach diagonal links vorn

Werte von 128 bis 136 bedeuten, daß zusätzlich der Feuerknopf gedrückt wurde.

Beispiel:

```
PRINT JOY(2)
135
READY.
```

Joystick 2 steht bei gedrücktem Feuerknopf in der Stellung nach links.

KEY–Befehl

Format: **KEY** [*n,x$*]

Zweck: Belegt Funktionstasten neu oder zeigt deren Belegung an.

n Beliebiger numerischer Ausdruck, dess en ganzzahliger Wert zwischen 1 und 8 liegen muß und der eine Funktionstaste bezeichnet.

x$ Beliebiger Zeichenkettenausdruck, dessen Wert der gewählten Funktiionstaste zugeordnet wird und der bei Betätigen dieser Taste auf dem Bildschirm angezeigt wird.

Bemerkungen: Wird der KEY–Befehl ohne Parameter gegeben, so werden die augenblicklichen Belegungen auf dem Bildschirm angezeigt.

Die Länge aller 8 Belegungen zusammen darf 256 Zeichen nicht überschreiten.

Beispiel:

```
KEY 8,"GRAPHIC 0"+CHR$(13)+"LIST"+CHR$(13)
```

Wenn Funktionstaste f8 (die vierte Taste zusammen mit SHIFT) im Direktmodus gedrückt wird, wird der Textmodus eingeschaltet und das im Hauptspeicher befindliche Programm wird gelistet.

LOCATE–Anweisung

Format: **LOCATE** ***x,y***

Zweck: Positioniert den grafischen Cursor (Pixel–Cursor) auf dem Bildschirm.

x,y Skalierte Koordinaten (s.a. SCALE–Anweisung). Für ***x*** (horizontale Koordinate) und für ***y*** (vertikale Koordinate) dürfen je nach Skalierung Werte zwischen 0 und 1023 angegeben werden.

Bemerkungen: Im Gegensatz zum normalen Text–Cursor ist der grafische Cursor nicht sichtbar.

Die aktuelle Position des grafischen Cursors ist für alle grafischen Zeichenanweisungen die Ausgangsposition.

Nach der Ausführung einer grafischen Zeichenanweisung ist der letzte gezeichnete Bildpunkt die neue aktuelle Position des grafischen Cursors.

Die aktuelle Position des grafischen Cursors kann mit Hilfe der RDOT–Funktion (s. dort) gefunden werden.

Beispiel:

```
LOCATE 160,100
```

Der grafische Cursor wird auf die Mitte des Bildschirms (hohe Auflösung, keine Skalierung) positioniert.

MONITOR–Befehl

Format: **MONITOR**

Zweck: Der Maschinensprache–Monitor wird aufgerufen.

Bemerkungen: Funktionen und Bedienung des Monitors sind ausführlich im Anhang C beschrieben.

MOVSPR–Anweisung

Format:

MOVSPR ***n,x,y***
MOVSPR ***n,+/–x1,+/–y1***
MOVSPR ***n,Winkel#Geschw***

Zweck:

Bewegt ein Sprite mit einer spezifizierten Geschwindigkeit oder plaziert ein Sprite an einer bestimmten Stelle auf dem Bildschirm.

n Ein ganzzahliger Wert zwischen 0 und 7, der die Nummer für das Sprite angibt.

x,y Die absoluten, skalierten Koordinaten, bei denen das Sprite positioniert werden soll.

+/–x1,+/–y1 Die zur letzten Sprite–Positionierung relativen, skalierten Koordinaten, bei denen das Sprite positioniert werden soll.

Winkel Ein ganzzahliger Wert zwischen 0 und 360, der angibt, in welchem Winkel das Sprite von seiner ursprünglichen Position wegbewegt werden soll.

Geschw Ein ganzzahliger Wert zwischen 0 und 15, der die Geschwindigkeit angibt, mit der das Sprite bewegt werden soll.

Bemerkungen:

Die Anwendung der MOVSPR–Anweisung ist ausführlich und an vielen Beispielen in Kapitel 4.7 beschrieben.

PAINT–Anweisung

Format: **PAINT** [***Farbquelle***][,[*x,y*][,***Modus***]]

Zweck: Malt einen Bildschirmbereich mit einer wählbaren Farbe aus.

Farbquelle Ein ganzzahliger Wert zwischen 0 und 3, der die Farbe definiert, mit der der Bereich ausgemalt werden soll. Voreingestellt ist die Vordergrundfarbe (1). Es bedeuten:

0 Hintergrundfarbe
1 Vordergrund– (Zeichen–)Farbe
2 Zusatzfarbe 1
3 Zusatzfarbe 2

x,y Koordinaten (skaliert) des Startpunktes zum Ausmalen. Voreingestellt ist hier die aktuelle Position des grafischen Cursors (Pixel–Cursor).

Modus Ein ganzzahligert Wert von 0 oder 1, der festlegt, ob der auszumalende Bereich von der gewählten Farbe (0) oder einer anderen Farbe als der Hintergrundfarbe umgeben ist (1).

Bemerkungen: Die PAINT–Anweisung füllt solange den Bereich um den angegebenen Punkt, bis ein Rand in der angegebenen Farbe (Modus 0) oder einer anderen als der Hintergrundfarbe erreicht ist.

Die Endposition des grafischen Cursors ist das angegebene Koordinatenpaar ***x,y.***

Wenn der Startpunkt bereits die angegebene Farbe beim Modus 0 bzw. irgendeine von der Hintergrundfarbe verschiedene Farbe beim Modus 1 hat, hat der Befehl PAINT keine Wirkung.

Beispiel:

```
10 CIRCLE ,160,100,65,50
20 PAINT ,160,100
```

Der in Zeile 100 gezeichnete Kreis wird in der Vordergrundfarbe (Voreinstellung) ausgemalt.

PEN–Funktion

Format: *v*= **PEN(*n*)**

Zweck: Liefert den Zustand oder die Bildschirmkoordinaten des Lichtstiftes.

n Ein ganzzahliger Wert zwischen 0 und 4, der folgende Wirkung hat:

- 0 x–Koordinate der Lichtstiftposition auf dem grafischen Bildschirm (60 bis 320).
- 1 y–Koordinate der Lichtstiftposition auf dem grafischen Bildschirm (50 bis 200).
- 2 Zeichen–Spaltenposition des Lichtstiftes beim 80–Zeichen–Bildschirm.
- 3 Zeichen–Zeilenposition des Lichtstiftes beim 80–Zeichen–Bildschirm.
- 4 Es wird 1 übergeben, falls der Lichtstift seit der letzten Abfrage aktiviert wurde, andernfalls 0.

Bemerkungen: Wie bei den Sprite–Koordinaten sind die Lichtstiftkoordinaten nicht skaliert, sondern echte Koordinaten innerhalb des umrahmten Bildschirmbereiches.

Liegt die Lichtstiftposition außerhalb des gültigen Bildschirmbereiches, sind die übergebenen Koordinatenwerte 0.

Für die PEN–Funktion braucht die COLLISION–Anweisung nicht aktiviert zu werden.

Im allgemeinen ist für die Stimulierung des Lichtstiftes ein weißer Bildschirmhintergrund erforderlich.

Die von der PEN–Funktion übergebenen Werte können von Bildschirm zu Bildschirm unterschiedlich sein.

Beispiel:

```
100 DO UNTIL PEN(4):LOOP
110 PRINT PEN(2);PEN(3)
```

Sobald der Lichtstift aktiviert wird, wird die Spalten– und Zeilen–Position des Bildschirmzeichens gedruckt, bei dem er aufgesetzt wurde.

π–Funktion

Format: *V*= π

Zweck: Übergibt den Wert der Ludolfschen Zahl (3.14159265).

Beispiel:

```
PRINT COS(π)
-1
READY.
```

PLAY–Anweisung

Format: **PLAY** ***Zeichenkette***

Zweck: Spielt die mit Hilfe einer Zeichenkette definierte Tonfolge.

Zeichenkette Eine Zeichenkette in Anführungszeichen, die sich aus Tonerzeugungs– und –abspielbefehlen zusammensetzt. Es gibt folgende Befehle:

O*n* Setzt eine von 7 Oktaven (*n*=0 bis *n*=6).

T*n* Setzt eine der folgenden 10 Klanghüllkurven:
- 0 Klavier (voreingestellt)
- 1 Akkordeon
- 2 Zirkusorgel
- 3 Trommel
- 4 Flöte
- 5 Gitarre
- 6 Cembalo
- 7 Orgel
- 8 Trompete
- 9 Xylophon

U*n* Setzt die Lautstärke (*n* von 0 bis 9)

V*n* Setzt eine von 3 möglichen Stimmen.

X*n* Schaltet das mit der FILTER–Anweisung (s. dort) gewählte Filter ein (*n*=1) oder aus (*n*=0).

N Eine der Noten A B C D E F G.

#*N* Es wird die Note ***N*** einen Halbton höher gespielt.

$*N* Es wird die Note ***N*** einen Halbton tiefer gespielt.

W Die folgende Note wird als ganze Note gespielt.

H Die folgende Note wird als halbe Note gespielt.

Q Die folgende Note wird als viertel Note gespielt.

I Die folgende Note wird als achtel Note gespielt.

S Die folgende Note wird als sechzehntel Note gespielt.

.*N* Die folgende Note wird als punktierte Note (die Hälfte ihres Wertes) gespielt.

R Setzt eine Pause.

M Wartet, bis die gegenwärtig gespielte Musik zuende ist.

POINTER–Funktion

Format: *v*= **POINTER(*Variable*)**

Zweck: Liefert die Hauptspeicheradresse des ersten Bytes eines Variablenzeigers in der Speicherbank 1, in der der BASIC-Interpreter alle Variablen, Felder und Zeichenketten ablegt.

Variable Eine beliebige Variable oder ein beliebiges Feldvariablenelement, der/dem bereits ein Wert zugewiesen sein muß.

Bemerkungen: Die übergebene Adresse im Bereich 0 bis 65535 stellt einen Offset in der Speicherbank 1 des Interpreters dar.

Wird POINTER für ein Feldvariablenelement verwendet, sollten alle einfachen Variablen vorher definiert sein, damit sich die Feldelementadressen nicht bei jeder neuen Definition einer einfachen Variablen wieder verändern.

POINTER ist nützlich, um die Anfangsadressen für Variablendeskriptoren an Maschinensprache–Unterprogramme zu übergeben.

Beispiel:

```
100 BX%=1024
110 VP=POINTER(BX%)
120 BANK 1:AX=256*PEEK(VP)+PEEK(VP+1)
```

Die Daten, die in der Ganzzahlvariablen BX% gespeichert sind, werden in Zeile 120 gelesen.

POT–Funktion

Format: ***v*= POT(*n*)**

Zweck: Liefert die Position eines der 4 anschließbaren Drehregler (paddles).

n Ein ganzzahliger Wert zwischen 1 und 4 der den Drehregler definiert.

1 Drehregler 1
2 Drehregler 2
3 Drehregler 3
4 Drehregler 4

Bemerkungen: Der von der POT–Funktion übergebene Wert kann zwischen 0 und 255 liegen. Ein um 256 erhöhter Wert bedeutet, daß außerdem der Feuerknopf gedrückt wurde.

PRINT USING–Anweisung

Format: **PRINT** [#***Filenr***] **USING** ***v$***,
Liste von Ausdrücken[;]

Zweck: Gibt die Werte von ***Liste von Ausdrücken*** auf dem Bildschirm oder einem Ausgabegerät formatiert aus.

Filenr Die Nummer, unter der eine Datei für ein Ausgabegerät eröffnet wurde. Bei Nummern zwischen 1 und 127 wird den ausgegebenen Daten ein Wagenrücklaufcode (CHR$(13)) angefügt. Bei Nummern zwischen 128 und 255 zusätzlich noch ein Zeilenvorschubcode (CHR$ (10)).

v$ Eine Zeichenkettenvariable oder –konstante, die das Druckformat definiert (s. Beispiele).

Liste von Ausdrücken Eine beliebige Liste von numerischen und/oder Zeichenkettenausdrücken, deren Werte ausgegeben werden sollen. Die einzelnen Ausdrücke müssen durch Komma (,), Semikolon (;) oder eine oder mehrere Leerstellen voneinander getrennt sein. Die Formatierung der Ausgabe erfolgt durch Steuerzeichen, deren Wirkung im folgenden getrennt für Zeichenketten– und numerische Daten anhand von Beispielen beschrieben wird. Hier führt das Komma nach einem Ausdruck jedoch **nicht** zur Tabulierung auf den Anfang des nächsten 10 Zeichen langen Druckfeldes.

Zeichenketten–Druckfelder

Jedes Nummernzeichen im Formatfeld steht für ein Zeichen im Druckfeld. Jedes Formatfeld muß mindestens ein Nummernzeichen enthalten. Hat der Wert eines Zeichenkettenausdruckes mehr Zeichen als Druckstellen mit # definiert sind, wird die Zeichenkette abgeschnitten, andernfalls wird mit Leerzeichen aufgefüllt. Die Daten werden linksbündig im Feld gedruckt.

= Die normalerweise linksbündig im Druckfeld angezeigten Daten werden zentriert angezeigt.

> Die Daten werden rechtsbündig im Druckfeld angezeigt.

Numerische Druckfelder

Hat der Wert eines numerisch Ausdruckes mehr Druckstellen als Positionen mit # definiert sind, wird das Druckfeld mit Sternen (*) gefüllt, hat er weniger, werden links Leerstellen aufgefüllt, da die Daten rechtsbündig gedruckt werden.

\+ und – Plus– oder Minus–Zeichen können entweder an erster oder letzter Stelle der Formatzeichenkette stehen. Das Vorzeichen des auszugebenden Wertes wird dann an der Stelle gedruckt.

. Legt die Dezimalpunktposition in einer Zahl mit Dezimalstellen fest. In jeder Formatzeichenkette darf nur ein Dezimalpunkt vorkommen.

, Legt die Position fest, an der im Druckfeld ein Komma ausgegeben werden soll. Dem Komma muß mindestens ein #–Symbol in der Formatzeichenkette vorangestellt sein. Das Komma ist **kein** Dezimalkomma, sondern dient nur der besseren Lesbarkeit von grossen Zahlen.

$ Immer vor der ersten gültigen Ziffer der auszugebenden Zahl wird ein Dollar–Zeichen gedruckt, wenn vor dem Dollar–Zeichen in der Formatkette ein #–Symbol steht.

↑↑↑↑ Wenn die Formatzeichenkette mit vier ↑–Symbolen endet, denen außerdem noch ein + oder – folgt, wird die darzustellende Zahl im Exponentialformat gedruckt.

Die folgende Tabelle enthält eine Reihe von Beispielen für die Verwendung der PRINT USING–Anweisung:

Format	Ausdruck	Ergebnis	Bemerkungen
+##	1	+ 1	Leerstelle zwischen Vorzeichen und Zahl
#.##+	–.01	0.01–	Führende Null hinzugefügt
–.##	–.1	–.10	führende Null durch Minus überschrieben
##.#–	1	1.0	Folgende Null hinzugefügt
+##+	1	SYNTAX ERROR	Zwei Plus–Symbole
+##.#–	1	SYNTAX ERROR	Plus– und Minus–Symbole
####	–100.5	–101	Auf insgesamt 4 Stellen gerundet
####	–1000	****	Überlauf, weil Feld nur für 4 Zeichen
#.##	–4E–03	–.00	Auf insgesamt 4 Stellen gerundet
##.	10	10.	Dezimalpunkt hinzugefügt
#.#.	1	SYNTAX ERROR	Zwei Dezimalpunkte
###,##	100	1,00	Kein Dezimalkomma, wie im Deutschen
##,##	10.4	10	Gerundet und Komma unterdrückt
#,###.##	10000.009	1,000.01	Gerundet
##,##	–1	–1	Komma unterdrückt
##,##	–10	–10	Komma wird von Minus überschrieben
##,##	–100.9	–1,01	Gerundet
##=>.>	1000	1000.0	= und < werden als # interpretiert, weil sie im Zahlenfeld stehen
+>==,#	1	+>==,1	Vor dem Komma muß mindestens ein # stehen, sonst werden >, = und Komma als druckbare Zeichen interpretiert
+>=#,#	1	+ 1	> und = werden als # interpretiert, weil sie im Zahlenfeld stehen
#$##	1	$1	Führendes $–Zeichen
###$	–1	–$1	Vorzeichen steht vor dem $
##$##	–1	–$1	Vorzeichen steht vor dem $
###$–	–1	$1–	Vorzeichen steht am Ende

Format	Ausdruck	Ergebnis	Bemerkung
+$###	–1	+$–1	Vor dem $ muß mindestens ein # stehen, sonst wird + als druckbares Zeichen interpretiert
+#$##	–1	–$1	
#.#$#	1	*****	Überlauf, weil kein Platz für $ vorhanden
+#.#↑↑↑↑	1	+1.0E+00	Ausdruck in Exponentialdarstellung
#↑↑↑↑+			
	–340	3E+02–	Vorzeichen hinter der Zahl
#↑↑↑↑	1.5E+11	2E+11	Gerundet
#↑↑↑↑	–1.5E+11	*****	Überlauf, weil kein Platz für das Vorzeichen
##.##↑↑↑↑	100000	10.00E+04	
##.##↑↑↑↑	–100000	–1.00E+05	
##↑↑↑	1	SYNTAX ERROR	Nur 3 Pfeile
##↑↑↑↑↑	34	34E+00↑	Der fünfte Pfeil wird als druckbares Zeichen interpretiert
##.##	CBM	CBM	Linksbündig gedruckter Text
###>##	CBM	CBM	Rechtsbündig in einem 5–Zeichen–Druckfeld
#####	CBM	CBM	Linksbündig in einem 5–Zeichen–Druckfeld
=####	CBM	CBM	Zentriert in einem 5–Zeichen-Druckfeld
#,$#=+	CBM	CBM	Wie oben, weil die anderen Zeichen als Druckzeichen interpretiert werden

PUDEF–Anweisung

Format: **PUDEF *v$***

Zweck: Erlaubt die Umdefinition von bis zu vier verschiedenen Steuerzeichen in der PRINT USING–Anweisung.

v$ Eine Zeichenkettenkonstante in Anführungszeichen (") oder –variable, deren Wert bis zu 4 Zeichen lang sein darf. Die Position dieser Zeichen in der Zeichenkette steht für ein bestimmtes Steuerzeichen der PRINT USING–Anweisung. Es gilt:

Pos. 1 Füllzeichen. Voreingestellt ist hier das Leerzeichen.

Pos. 2 Komma. Voreingestellt ist ebenfalls das Komma.

Pos. 3 Dezimalpunkt. Voreingestellt ist ebenfalls der Dezimalpunkt.

Pos. 4 Dollarzeichen. Voreingestellt ist ebenfalls das Dollarzeichen.

Bemerkungen: PUDEF ändert die Zeichen nur, wenn eine PRINT USING-Anweisung zur Ausgabe verwendet wird. Bei PRINT hat PUDEF keine Wirkung.

PUDEF beeinflußt nur die Ausgabe selbst, nicht die Formatzeichenkette von PRINT USING.

Beispiel 1:

```
PUDEF "?"
```

Leerstellen werden beim Ausdruck durch Fragezeichen ersetzt.

Beispiel 2:

```
10 PUDEF ".,"
20 PRINT USING "###,###.##";-1234.567
RUN
-1.234,57
READY.
```

Das Komma in der Tausenderposition und der Dezimalpunkt (amerikanische Schreibweise) werden in die deutsche Schreibweise umgewandelt.

RCLR–Funktion

Format: *v*= **RCLR(*n*)**

Zweck: Liefert den der angegebenen Farbquelle zugeordneten aktuellen Farbcode im Bereich zwischen 1 und 16 (s.a. COLOR–Anweisung).

n Ein ganzzahliger Wert zwischen 0 und 6, der die Farbquelle angibt. Folgende Farbquellen sind verfügbar:

0 Hintergrund (40–Zeichen–Anzeige)
1 grafischer Vordergrund
2 Mehrfarbenmodus 1
3 Mehrfarbenmodus 2
4 Rand (40–Zeichen–Anzeige)
5 Text (80–Zeichen–Anzeige)
6 Hintergrund (80–Zeichen–Anzeige)

Beispiel:

```
A=RCLR(4)
```

Der aktuelle Farbcode für den Bildschirmrand wird der Variablen A zugewiesen.

RDOT–Funktion

Format: ***v*= RDOT(*n*)**

Zweck: Liefert die aktuelle Bildschirm–Position des grafischen Cursors (Pixel–Cursor) oder den für die aktuelle Position gültigen Code der Farbquelle.

n ein ganzzahliger Wert zwischen 0 und 3 mit folgender Bedeutung:

0 x–Position
1 y–Position
2 Farbquell–Code bei aktueller Position

Beispiel:

```
10 GRAPHIC 1,1
20 CIRCLE 1,160,100,80,80
30 DRAW TO RDOT(0),180
```

Zeichnet einen Kreis mit senkrechtem Durchmesser.

RECORD–Anweisung

Format: **RECORD #*Filenr,Satznr*[*,Bytenr*]**

Zweck: Positioniert den Schreib/Lese–Zeiger auf eine wählbare Position innerhalb einer Disketten–Relativ–Datei (s.a. Kapitel 6.7).

Filenr Die logische Nummer im Bereich zwischen 1 und 255, unter der die Disketten–Relativ–Datei eröffnet wurde.

Satznr Ein ganzzahliger Wert zwischen 1 und 65535, der den logischen Datensatz in der Diskettendatei bezeichnet, auf den zugegriffen werden soll.

Bytenr Ein ganzzahliger Wert zwischen 1 und 254, der das Byte im gewünschten logischen Datensatz der Relativ–Datei bezeichnet, auf den der Schreib/Lesezeiger gestellt werden soll. Voreingestellt ist hier 1.

Bemerkungen: Werden für die Parameter Variablen verwendet, so müssen diese in Klammern eingekleidet werden.

Positionieren auf nichtexistente Sätze führt zu der Fehlermeldung RECORD NOT PRESENT.

Beispiele:

```
RECORD #2,452,23
```

Der Schreiblesezeiger für die unter der logischen Dateinummer 2 eröffneten Relativ–Datei wird auf das 23. Byte im 452. Satz gestellt.

RENAME–Befehl

Format: **RENAME *Altname* TO *Neuname*[,D*Laufw*]**
[,U*Geräteadr*]

Zweck: Weist einer Diskettendatei auf einem wählbaren Laufwerk einen neuen Namen zu (s.a. Kapitel 6.9.4).

Altname,Neuname Zeichenkettenkonstanten in Anführungszeichen (") oder –variablen mit einer Länge von bis zu 16 Zeichen, die den alten bzw. neuen Namen für die Diskettendatei bezeichnen.

Laufw Ein ganzzahliger Wert von 0 oder 1, der das Laufwerk bezeichnet. Voreingestellt ist hier 0.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15 für die Geräteadresse der angeschlossenen Floppy–Disk. Voreingestellt ist hier 8.

Bemerkungen: Werden für die Parameter Variablen verwendet, so müssen diese in Klammern angegeben werden.

Beispiele:

```
RENAME "ADRESSEN" TO "ADRNEU"
```

Die Datei ADRESSEN auf der Diskette in Laufwerk 0 der Floppy–Disk 8 erhält den Namen ADRNEU.

```
RENAME (AD$) TO (ND$),D(LW),U(GA)
```

Hier werden für die Parameter Variablen verwendet.

RENUMBER–Befehl

Format: **RENUMBER** [*Nnum*[,[*Schrittw*][,*Anum*]]]

Zweck: Numeriert das im Hauptspeicher befindliche BASIC–Programm vollständig oder teilweise neu.

Nnum Die Zeilennummer, mit der der neu zu numerierende Programmteil beginnen soll. Voreingestellt ist hier 10.

Schrittw Ein ganzzahliger Wert, um den bei der Neunumerierung jede Zeilennummer gegenüber der vorhergehenden steigen soll. Voreingestellt ist hier 10.

Anum Die Zeilennummer im aktuellen Programm, von der an neu numeriert werden soll. Voreingestellt ist hier 0.

Bemerkungen: Dieser Befehl kann nur im Direktmodus gegeben werden.

Alle Zeilenreferenzen in den Anweisungen

GOTO
GOSUB
THEN
ELSE
ON GOTO . . .
ON GOSUB . . .
RESTORE
RESUME
EL

werden durch RENUMBER der neuen Numerierung angepaßt.

RENUMBER kann die Reihenfolge von Programmzeilen nicht verändern.

Beispiele:

```
RENUMBER
```

Das gesamte im Haupspeicher befindliche Programm wird in 10er–Schritten neu numeriert. Die erste Zeile erhält die Nummer 10.

```
RENUMBER 100,15
```

Das gesamte im Hauptspeicher befindliche Programm wird in 15er–Schritten neu numeriert. Die erste Zeile erhält die Nummer 100.

```
RENUMBER 2500,50,1000
```

Das im Hauptspeicher befindliche Programm wird von Zeile 1000 an in 50er–Schritten neu numeriert. Zeile 1000 erhält die Nummer 2500.

RESTORE–Anweisung

Format: **RESTORE** [*Zeilennr*]

Zweck: Setzt den Lesezeiger der READ–Anweisung (s. dort in Kapitel 3) auf das erste DATA–Element der ersten oder einer wählbaren DATA–Zeile im Programm, um DATA-Elemente wiederholt zu lesen.

Zeilennr Eine gültige Programmzeilennummer, bei der eine DATA–Anweisung stehen muß.

Beispiel:

```
100 DATA 1,2,3,4,5,6
110 READ U,V,W:RESTORE 100
120 READ X,Y,Z
130 PRINT U;V;W;X;Y;Z
RUN
1 2 3 1 2 3
READY.
```

RESUME–Anweisung

Format:

RESUME
RESUME NEXT
RESUME *Zeilennr*

Zweck:

Setzt das Programm nach Ausführen einer Fehlerbehandlungsroutine (s. TRAP–Anweisung) an einer definierten Stelle fort.

RESUME Das unterbrochene Programm wird mit der Anweisung, die den Fehler verursacht hat, fortgesetzt.

RESUME NEXT Das unterbrochene Programm wird mit der Anweisung, die der fehlerverursachenden Anweisung folgt, fortgesetzt.

RESUME *Zeilennr* Das unterbrochene Programm wird mit der durch ***Zeilennr*** definierten Zeile fortgesetzt.

Beispiel:

```
100 TRAP 500
.
.
.
500 IF ER=11 THEN RESUME 250
```

Bei einem SYNTAX ERROR soll das Programm mit Zeile 250 fortgesetzt werden.

RGR–Funktion

Format: ***v*=RGR(*n*)**

Zweck: Liefert den aktuell eingestellten grafischen Modus (0 bis 5; s. a. GRAPHIC–Anweisung).

n Ein Scheinargument, dessen Wert beliebig sein kann, aber angegeben werden muß.

RREG–Anweisung

Format: **RREG** [*AVar*][,[*XVar*][,[*YVar*]
[,[**SVar**]]]

Zweck: Weist den spezifizierten Variablen die bei der Beendigung eines mit der SYS–Anweisung (s. dort in Kapitel 5.2) aufgerufenen Maschinensprache–Unterprogramms aktuellen Inhalte der Prozessorregister A, X, Y und SR (Statusregister) zu.

AVar,XVar,YVar,SVar Beliebige numerische Variablen oder Feldelemente.

Bemerkungen: Mit dieser Anweisung können die Prozessorregisterinhalte nach Beendigung eines Maschinensprache–Unterprogramms in ein BASIC–Programm übernommen werden.

RSPCOLOR–Funktion

Format: ***v*= RSPCOLOR(*n*)**

Zweck: Liefert den Code der aktuellen Zusatzfarbe für Sprites.

n Ein ganzzahliger Wert von 1 oder 2 mit folgender Bedeutung:

1 liefert einen Farbcode zwischen 1 und 16 für die Sprite–Zusatzfarbe 1.

2 liefert einen Farbcode zwischen 1 und 16 für die Sprite–Zusatzfarbe 2.

RSPPOS–Funktion

Format: ***v*= RSPPOS(*n*,*m*)**

Zweck: Liefert für ein bestimmtes Sprite Position oder aktuelle Geschwindigkeit.

n Ein ganzzahliger Wert zwischen 1 und 8, der das zu untersuchende Sprite definiert.

m Ein ganzzahliger Wert zwischen 0 und 2 mit folgender Bedeutung:

0 liefert aktuelle x–Koordinate
1 liefert aktuelle y–Koordinate
2 liefert die aktuelle Geschwindigkeit im Bereich zwischen 0 und 15

RSPRITE–Funktion

Format: ***v*= RSPRITE(*n*,*m*)**

Zweck: Liefert für ein bestimmtes Sprite die aktuellen Attribute.

n Ein ganzzahliger Wert zwischen 1 und 8, der das zu untersuchende Sprite definiert.

m Ein ganzzahliger Wert zwischen 0 und 5 mit folgender Bedeutung:

0 Liefert 1, wenn das Sprite aktiviert und 0, wenn es inaktiviert ist.
1 Liefert die Spritefarbe im Bereich 1 bis 16.
2 Liefert eine 0, wenn das Sprite Priorität über den Hintergrund hat, andernfalls 1.
3 Liefert eine 1, wenn das Sprite in x–Richtung gedehnt ist, andernfalls 0.
4 Liefert eine 1, wenn das Sprite in y–Richtung gedehnt ist, andernfalls 0.
5 Liefert eine 1, wenn für das Sprite der Mehrfarbenmodus aktiv ist, andernfalls 0.

RUN–Befehl

Format: **RUN** [***Zeilennr***]
RUN ***Dateiname***[,**D** ***Laufw***][,**U** ***Geräteadr***]

Bedeutung: Ein BASIC–Programm, das entweder im Hauptspeicher resident ist oder von Disk geladen wird, wird gestartet.

Zeilennr Eine gültige Programmzeilennummer, bei der das Programm gestartet werden soll.

Dateiname Eine Zeichenkettenkonstante in Anführungsstrichen (") oder eine –Variable in Klammern (), die den Programmdateinamen bezeichenet.

Laufw Ein ganzzahliger Wert von 0 oder 1. Voreingestellt ist hier 0.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15, der die Geräteadresse der angeschlossenen Floppy-Disk bezeichnet. Voreingestellt ist hier der Wert 8.

Bemerkungen: Wird keine Zeilennummer angegeben, wird das Programm mit der niedrigsten Zeilennummer gestartet.

Durch RUN werden vor dem Programmstart alle ggfs. definierten Variablen gelöscht.

Wird im RUN–Befehl eine Datei spezifiziert, so wird vor dem Laden der Programmspeicher gelöscht. Außerdem werden alle ggf. geöffneten Dateien geschlossen.

Beispiel 1:

```
100 FOR I=1 TO 2:PRINT I↑2;:NEXT
200 PRINT "HALLO!"
RUN
1 4HALLO!
READY.
RUN 200
HALLO!
READY.
```

Beispiel 2:

```
RUN "PROG1",D0,U9
```

Das Programm PROG1 wird von der Diskette in Laufwerk 0 der Floppy–Disk mit der Geräteadresse 9 geladen und gestartet.

RWINDOW–Funktion

Format: ***v*= RWINDOW(*n*)**

Zweck: Liefert die Parameter für das aktuelle Bildfenster (s.a. WINDOW–Befehl).

n Ein ganzzahliger Wert zwischen 0 und 2 mit folgender Bedeutung:

0 Liefert die Nummer der letzten Zeile im aktuellen Bildfenster (0–24).
1 Liefert die Nummer der letzten Spalte im aktuellen Bildfenster (0–79).
2 Liefert 40 oder 80, je nach eingestellter Bildschirmbreite.

SCALE–Anweisung

Format: **SCALE** [*n*][,*xmax,ymax*]

Zweck: Schaltet die Skalierung im Grafikmodus ein oder aus und verändert den Maßstab für die Bildschirmkoordinaten.

n Ein ganzzahliger Wert von 0 oder 1 mit folgender Bedeutung:

0 schaltet die Skalierung aus. Die grafischen Bildschirmkoordinaten x/y sind
0–159/0–199 im Mehrfarbenmodus und
0–319/0–199 im hochauflösenden Modus.

1 schaltet die Skalierung ein. Die grafischen Bildschirmkoordinaten sind wie unten beschrieben eingestellt.

xmax,ymax Größte mögliche x– bzw. y–Koordinate bei eingeschalteter Skalierung. ***xmax*** muß zwischen 160 bzw. 320 und 1023 liegen.***ymax*** muß zwischen 200 und 1023 liegen. Voreingestellt ist für eingeschaltete Skalierung 1023 für beide Koordinaten.

Bemerkungen: Mit der SCALE–Anweisung lassen sich im grafischen Modus Objekte auf dem Bildschirm in anderen Maßstäben darstellen.

Beispiel:

```
SCALE 1
```

Die Bildschirmskalierung wird eingeschaltet und ggfs. abgebildete Objekte werden verkleinert, da jetzt die Maximalkoordinaten in x– und y–Richtung 1023 sind.

SCNCLR–Anweisung

Format: **SCNCLR** [*n*]

Zweck: Löscht den angegebenen Bildschirm.

n ein ganzzahliger Wert mit folgender Bedeutung:

0 Der 40–Zeichen–Textschirm wird gelöscht.

1 Der hochauflösende Grafikbildschirm wird gelöscht.

2 Der hochauflösende geteilte Bildschirm (Text– und Grafikdarstellung) wird gelöscht.

3 Der Mehrfarben–Grafikbildschirm wird gelöscht.

4 Der geteilte Mehrfarben–Grafikbildschirm (Text– und Grafikdarstellung) wird gelöscht.

5 Der 80–Zeichen–Textschirm wird gelöscht.

Bei aktiviertem Grafikbildschirm (s. GRAPHIC–Anweisung) kann SCNCLR ohne Parameter eingegeben werden.

SCRATCH–Befehl

Format: **SCRATCH** ***Dateiname***[,D***Laufw***]
[,U***Geräteadr***]

Zweck: Löscht den Dateieintrag für die angegebene Datei aus dem Diskettenverzeichnis und damit die Datei von Diskette (s.a. Kapitel 6.8).

Dateiname Eine Zeichenkettenkonstante in Anführungszeichen (") oder eine –variable mit einer Länge von bis zu 16 Zeichen, die die zu löschende Datei bezeichnet.

Laufw Ein ganzzahliger Wert von 0 oder 1, der das Laufwerk bezeichnet. Voreingestellt ist hier 0.

Geräteadr Ein ganzzahliger Wert zwischen 4 und 15, für die Geräteadresse der angeschlossenen Floppy–Disk. Voreingestellt ist hier 8.

Bemerkungen: Der SCRATCH–Befehl verlangt durch die Anfrage ARE YOU SURE? vom Anwender die Bestätigung für die Befehlsausführung (Antwort Y für YES). Bei jeder anderen Antwort wird der Befehl ignoriert.

Werden in dem Dateinamen die Joker–Zeichen * und ? (s. Kapitel 6.3) verwendet, so können mit einem Befehl mehrere Dateien gelöscht werden.

Wird der SCRATCH–Befehl im Direktmodus gegeben, so meldet er die erfolgreiche Löschung mit

01, FILES SCRATCHED,nn,00

wobei nn die Anzahl der gelöschten Dateien angibt.

Beispiele:

```
SCRATCH"DATEN"
```

Die Datei "DATEN" wird auf der Diskette in Laufwerk 0 von der Floppy–Disk mit der Geräteadresse 8 gelöscht.

```
SCRATCH"*",U9
```

Alle Dateien werden von der Diskette in Laufwerk 0 von Floppy–Disk Nr. 9 gelöscht.

```
SCRATCH"????.PRG",D1
```

Alle Dateien, deren 8 Zeichen lange Namen mit .PRG enden, werden von der Diskette in Laufwerk 1 (bei Doppellaufwerken) von Floppy–Disk Nr. 8 gelöscht.

```
SCRATCH(A$),D(LW),U(GA)
```

Hier werden für alle Parameter Variablen benutzt.

SLEEP–Befehl

Format: **SLEEP *n***

Zweck: Hält die Programmausführung für eine bestimmte Zeit an.

n Ein ganzzahliger Wert zwischen 1 und 65535, der die Wartezeit in Sekunden definiert.

Bemerkungen: Die durch den SLEEP–Befehl gesetzte Zeit wird durch die FAST–Anweisung (s. dort) nicht beeinflußt.

SLOW–Befehl

Format: **SLOW**

Zweck: Schaltet den Prozessor vom Betrieb im 2–Megahertz-Takt auf Betrieb im 1–MegahertzTakt um. Der Rechner wird damit halb so schnell.

Bemerkungen: Die 40–Zeichen–Bildschirmanzeige wird durch den SLOW–Befehl wieder aktiviert (s.a. FAST–Befehl).

SOUND–Anweisung

Format: **SOUND *Stimme,Freq,Dauer***[,[***Richtung***]
[,[***Maxfreq***][,[***Stufe***][,[***Welle***]
[,***Impulsbreite***]]]]]

Zweck: Erzeugt Ton– oder Klangeffekte wählbarer Frequenz und Dauer und gibt sie über den Lautsprecher des Monitors oder Fernsehempfängers aus.

Stimme Ein ganzzahliger Wert zwischen 1 und 3 mit folgender Bedeutung:

1 Stimme 1 (Ton)
2 Stimme 2 (Ton)
3 Stimme 3 (Ton)

Freq Ein ganzzahliger Wert zwischen 0 und 65535 für die Frequenz.

Dauer Ein ganzzahliger Wert zwischen 0 und 32767, der die Dauer des Klangeffektes in Rechnerzeittakten (1/60 Sekunde) angibt. Ein Wert von 0 bricht alle ggfs. laufenden Ton– oder Klangeffekte mit sofortiger Wirkung ab.

Richtung Ein ganzzahliger Wert zwischen 0 und 2 mit folgender Bedeutung:

0 zunehmende Klangstufen
1 abnehmende Klangstufen
2 oszillierende Klangstufen

Voreingestellt ist hier 0.

Maxfreq Ein ganzzahliger Wert zwischen 0 und 65535 für die Maximalfrequenz für anschwellende Klangeffekte. Voreingestellt ist hier 0.

Stufe Ein ganzzahliger Wert zwischen 0 und 32767, der die Zeitdauer für stufige Klangeffekte angibt.

Welle Ein ganzzahliger Wert zwischen 0 und 3 für die Wellenform:

0 Dreieck
1 Sägezahn
2 Rechteck
3 Rauschen

Voreingestellt ist hier 2.

Impulsbreite Ein ganzzahliger Wert zwischen 0 und 4095. Voreingestellt ist hier 2048 (50 % Tastverhältnis).

Bemerkungen: Wenn eine SOUND–Anweisung mit einer bestimmten Stimme gegeben wird, solange eine vorher gegebene SOUND-Anweisung mit derselben Stimme noch aktiv ist, so wird erstere zunächst zuende geführt, ehe die zweite begonnen wird.

Beispiel:

```
SOUND 2,800,3600
```

Spielt eine Note der Stimme 2 bei einem Frequenzwert von 800 für eine Minute.

SPRCOLOR–Anweisung

Format: **SPRCOLOR** [***Zusatzfarb1***][,***Zusatzfarb2***]

Zweck: Setzt die Zusatzfarbe im Mehrfarbenmodus für alle Sprites.

Zusatzfarb1 Ein ganzzahliger Wert zwischen 1 und 16, der die Zusatzfarbe 1 für alle Sprites im Mehrfarbenmodus festlegt.

Zusatzfarb2 Ein ganzzahliger Wert zwischen 1 und 16, der die Zusatzfarbe 2 für alle Sprites im Mehrfarbenmodus festlegt.

SPRDEF–Befehl

Format: **SPRDEF**

Zweck: Schaltet den Sprite–Editor ein.

Bemerkungen: Dieser Befehl ist sowohl im Direktmodus als auch im Programmodus erlaubt. Der Sprite–Editor kennt folgende Befehle:

1 bis 8 Wählt eine Sprite–Nummer.

A Schaltet die automatische Wiederholung für die Zeichen–Tasten 1, 2, 3 oder 4 ein oder aus.

CRSR–Tasten Steuern den Cursor über das Sprite–Raster.

RETURN–Taste Setzt den Cursor auf die nächste Zeile des Sprite–Rasters.

SHIFT–RETURN Speichert das entworfene Sprite. Wird unmittelbar danach die RETURN–Taste gedrückt, so wird der Sprite–Editor beendet und der Interpreter wird wieder in den Direktmodus gesetzt.

HOME–Taste Setzt den Cursor in die linke obere Ecke des Sprite–Rasters.

CLR–Taste Löscht das gesamte Sprite–Raster.

1 bis 4 Im hochauflösenden Grafik–Modus zeichnet 1 einen Sprite–Bildpunkt in der aktuellen Hintergrundfarbe und 2 einen Sprite–Bildpunkt in der aktuellen Vordergrundfarbe bei der Cursorposition.
Im Mehrfarbenmodus haben die 1– und 2–Taste dieselbe Funktion wie im hochauflösenden Grafik–Modus. Die 3–Taste zeichnet einen Sprite–Bildpunkt in der Zusatzfarbe 1 und die 4–Taste einen in der Zusatzfarbe 2 des Mehrfarbenmodus. In diesem Fall werden immer doppelte Bildpunkte gezeichnet, da im Mehrfarbenmodus nur

die halbe Bildschirmauflösung in x–Richtung zur Verfügung steht.
Jeder im Raster gezeichnete Bildpunkt wird rechts in der Originaldarstellung der aktuellen Bildschirmauflösung wiederholt, um eine Kontrolle über die tatsächliche Größe des zu entwerfenden Sprites zu haben.

CTRL 1 bis CTRL 8 Wählt eine der 8 Vordergrundfarben 1 bis 8 für das Sprite.

C= 1 bis C= 8 (Commodore–Taste) Wählt eine der 8 Vordergrundfarben 9 bis 16.

STOP–Taste Löscht alle Sprite–Eingaben. Wird unmittelbar nach der STOP–Taste die RETURN–Taste gedrückt, so wird der Sprite–Editor beendet und der Direktmodus wird wieder aktiviert.

X Dehnt das Sprite in x–Richtung.

Y Dehnt das Sprite in y–Richtung.

M Wählt Vielfarbmodus 1 oder 2 für das Sprite.

Bemerkungen:

Wenn der Sprite–Editor aufgerufen wird, wird im linken oberen Teil des Bildschirms ein Raster von 21 x 24 Zeichenpositionen eingerichtet.

Um einen Bildpunkt zu zeichnen, wird der Sprite–Cursor (Plus–Zeichen) an die gewünschte Position im Sprite-Raster gesteuert und dann werden je nach eingestelltem grafischen Modus die Zifferntasten 1, 2, 3 oder 4 gedrückt.

Möchten Sie Ihre entworfenen Sprites auf Diskette speichern verfahren Sie im Direktmodus wie folgt:

BSAVE "*name*",B0,P3584 TO P4095

Zum Laden einer solchen Datei geben Sie nur ein:

BLOAD "*name*".

Mit Hilfe des Monitors müssen Sie zum Speichern auf Kassette oder Diskette

S"*name*",*gerät*,0E00,0FFF

und zum Laden

L"*name*",*gerät*

eingeben. Für ***name*** ist der Dateiname und für ***gerät*** die Geräteadresse in hexadezimaler Schreibweise anzugeben.

SPRITE–Anweisung

Format: **SPRITE** ***n***[,[***akt***][,[***Farbe***][,[***Pri***]
[,***xdehn***][,[***ydehn***]
[,[***Modus***]]]]]]

Zweck: Setzt Attribute für eines von 8 Sprites.

n Ein ganzzahliger Wert zwischen 1 und 8 für die Sprite-Nummer.

akt Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das Sprite aktiviert (1) oder inaktiviert (0) werden soll.

Farbe Ein ganzzahliger Wert zwischen 1 und 16 für die Sprite–Farbe. Die Zuordnung von Farbcodes und Farben ist bei der COLOR–Anweisung (s. dort) beschrieben.

Pri Ein ganzzahliger Wert von 0 oder 1, der festlegt, ob das Sprite bei Bewegung zwischen vorhandenen Bildschirmobjekten vor (0) oder hinter (1) diesen Objekten vorbeiläuft.

xdehn Ein ganzzahliger Wert von 0 oder 1 , der angibt, ob das Sprite in x–Richtung gedehnt ist (1) oder nicht (0).

ydehn Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das Sprite in y–Richtung gedehnt ist (1) oder nicht (0).

Modus Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das Sprite im hochauflösenden (0) oder im Mehrfarbenmodus angezeigt werden soll.

Bemerkungen: Nicht angegebene Parameter bedeuten, daß ggfs. vorher gesetzte Sprite–Attribute erhalten bleiben.

Der hochauflösende oder Mehrfarben–Sprite–Modus darf nicht mit den hochauflösenden bzw. Mehrfarben-Grafik-Modi verwechselt werden. Hochauflösende Spri-

tes können im Mehrfarben–Grafikmodus verwendet werden und umgekehrt.

Zur Ermittlung der für ein bestimmtes Sprite eingestellten Attribute kann die RSPRITE–Funktion (s. dort) verwendet werden.

Beispiel:

```
SPRITE4,1,6,1,,,1
```

Das Sprite Nr. 4 wird aktiviert. Es wird im Vielfarbmodus in der Farbe grün angezeigt und hat Priorität über angezeigte Bildschirmdaten.

SPRSAV–Anweisung

Format: **SPRSAV *n,v$***
SPRSAV *v$,n*

Zweck: Speichert ein bestimmtes Sprite als Punktraster in einer Zeichenkettenvariablen oder umgekehrt.

n Ein ganzzahliger Wert zwischen 1 und 8 für die Sprite-Nummer.

v$ Eine beliebige Zeichenkettenvariable, die das Punktraster als Binärinformation aufnimmt oder enthält.

Beispiele:

```
SPRSAV 1,A$
```

Überträgt das Punktraster von Sprite 1 als Binärinformation in die Zeichenkettenvariable A$.

```
SPRSAV B$,4
```

Überträgt den Wert der Zeichenkettenvariablen B als binäres Punktraster in das Sprite 4.

SSHAPE–Anweisung

Format: **SSHAPE *Zeikett,x1,y1*[*,x2,y2*]**

Zweck: Übeträgt den Inhalt eines angegebenen Bereiches auf dem Grafikbildschirm als binäre Bildinformation in eine Zeichenkettenvariable.

Zeikett Eine Zeichenkettenvariable, die den gewählten Bildschirmbereich als binäre Bildinformation aufnimmt.

x1,y1 Die skalierten Bildschirmkoordinaten der linken oberen Ecke des darzustellenden Bildes.

x2,y2 Die skalierten Bildschirmkoordinaten der rechten unteren Ecke des darzustellenden Bildes. Voreingestellt ist hier die aktuelle Position des grafischen Cursors (Pixel-Cursor).

Bemerkungen: Da BASIC–Zeichenketten auf eine Maximallänge von 255 Zeichen beschränkt sind, ist auch die Größe des zu speichernden grafischen Bereiches beschränkt.

Je nach aktiviertem grafischen Modus (hochauflösend oder Mehrfarben) läßt sich die benötigte Länge der Zeichenkette folgendermaßen berechnen:

L(m) = INT((ABS(*x1–x2*) + 1)/4 + .99) *
(ABS(*y1–y2*) + 1) + 4

L(h) = INT((ABS(*x1–x2*) + 1)/8 + .99) *
(ABS(*y1–y2*) + 1) + 4

STASH–Befehl

Format: **STASH *Bytes,Intadr,Extadr,Bank***

Zweck: Überträgt eine wählbare Anzahl von Bytes aus dem BASIC–Arbeitsspeicher in eine Speichererweiterungsbank.

Bytes Ein ganzzahliger Wert zwischen 0 und 65535, der die Anzahl der zu übertragenden Bytes angibt.

Intadr Ein ganzzahliger Wert zwischen 0 und 65535, der die Adresse im BASIC–Arbeitsspeicher angibt, ab der die Bytes umgespeichert werden sollen.

Extadr Ein ganzzahliger Wert zwischen 0 und 65535, der die Adresse in der Speicherbank angibt, ab der die Bytes abgelegt werden sollen.

Bank Ein ganzzahliger Wert zwischen 0 und 7, der die Speicherbank angibt, in der die Bytes abgelegt werden sollen.

Bemerkungen: Dieser Befehl kann nur ausgeführt werden, wenn eine Ram–Disk an den Rechner angeschlossen ist. Die Ram Disk decodiert den entsprechenden Befehl, und führt ihn aus.

Der Parameter Intadr bezieht sich auf die Bank, die mit dem BANK Befehl eingestellt wurde. Voreingestellt ist Bank 15.

Sollen z. B. Daten aus Bank 0 in der Ram Disk abgespeichert werden, so muß vor dem STASH Befehl 'BANK 0' eingegeben werden.

Beispiel:

```
STASH 1000,52000,2000,7
```

Überträgt 1000 Bytes ab Adresse 52000 aus dem BASIC-Arbeitsspeicher in die Speicherbank 7 ab Adresse 2000.

Achtung:
Um diesen Befehl ausführen zu können, brauchen Sie das Steckmodul Ram–Disk.

SWAP–Befehl

Format: **SWAP *Bytes,Intadr,Extadr,Bank***

Zweck: Tauscht eine wählbare Anzahl Bytes zwischen dem BASIC–Arbeitsspeicher und einer Speichererweiterungsbank.

Bytes Ein ganzzahliger Wert zwischen 0 und 65535, der die Anzahl der auszutauschenden Bytes angibt.

Intadr Ein ganzzahliger Wert zwischen 0 und 65535, der die Adresse im BASIC–Arbeitsspeicher angibt, ab der die Bytes ausgetauscht werden sollen.

Extadr Ein ganzzahliger Wert zwischen 0 und 65535, der die Adresse in der Speicherbank angibt, ab der die Bytes ausgetauscht werden sollen.

Bank Ein ganzzahliger Wert zwischen 0 und 7, der die Speicherbank angibt, mit der die Bytes ausgetauscht werden sollen.

Bemerkungen: Dieser Befehl kann nur ausgeführt werden, wenn eine Ram–Disk an den Rechner angeschlossen ist. Die Ram Disk decodiert den entsprechenden Befehl, und führt ihn aus.

Der Parameter Intadr bezieht sich auf die Bank, die mit dem BANK Befehl eingestellt wurde. Voreingestellt ist Bank 15.

Sollen z. B. Daten aus Bank 0 in der Ram Disk abgespeichert werden, so muß vor dem STASH Befehl 'BANK 0' eingegeben werden.

Beispiel:

```
SWAP 1000,52000,2000,7
```

Tauscht 1000 Bytes ab Adresse 52000 im BASIC–Arbeitsspeicher mit einem gleichgroßen Bereich in der Speicherbank 7 ab Adresse 2000 aus.

Achtung:
Um diesen Befehl ausführen zu können, brauchen Sie das Steckmodul Ram–Disk.

TEMPO–Anweisung

Format: **TEMPO *n***

Zweck: Setzt das Spieltempo für Musiknoten.

n Ein ganzzahliger Wert zwischen 0 und 255, der die relative Dauer angibt, mit der die gespielten Musiknoten gehalten werden. Die tatsächliche Dauer berechnet sich nach

Dauer $= 19.22/n$ in Sekunden.

Voreingestellt ist der Wert 8. Bei einem Wert von 0 wird die Note dauernd gehalten.

TRAP–Anweisung

Format: **TRAP** [*Zeilennr*]

Zweck: Die Fehlerunterbrechung wird aktiviert und das Programm verzweigt zu einer angegebenen Zeilennummer, sobald der Interpreter eine Fehlermeldung ausgeben will.

Zeilennr Eine gültige Programmzeilennummer, mit der die Fehlerbehandlungsroutine beginnt. Wird keine Zeilennummer angegeben, so wird die Fehlerunterbrechung inaktiviert.

Bemerkungen: Bei aktivierter Fehlerunterbrechung verzweigt das Programm bei allen Fehlerbedingungen einschließlich der Betätigung der STOP–Taste.

Beim Auftreten eines Fehlers wird eine Fehlermarke gesetzt, das Programm verzweigt zu der angegebenen Zeilennummer und führt dort eine Fehlerbearbeitung aus.

Die Nummer der fehlerhaften Zeile kann mit Hilfe der EL–Systemvariablen (s. dort), der Fehlercode mit Hilfe der ER–Systemvariablen und der Fehlermeldungstext mit der Funktion ERR$(ER) ermittelt werden.

Mit Hilfe der RESUME–Anweisung (s. dort) wird das Hauptprogramm fortgesetzt.

Achtung
Ein Fehler innerhalb einer Fehlerbearbeitungsroutine kann nicht mit TRAP aufgefangen werden.

TRON– und TROFF–Befehle

Format: **TRON**
TROFF

Zweck: Schaltet die Programmablaufverfolgung ein (TRON) oder aus (TROFF).

Bemerkungen: Bei eingeschalteter Programmablaufverfolgung werden die Zeilennummern in der Reihenfolge ihrer Bearbeitung durch den Interpreter in eckigen Klammern ([ ]) auf dem Bildschirm angezeigt.

Die Befehle TRON und TROFF dienen zum schrittweisen Austesten von BASIC–Programmen.

VOL–Anweisung

Format: **VOL *n***

Zweck: Setzt die Lautstärke für die mit der SOUND–und der PLAY–Anweisung (s. dort) erzeugten Ton– oder Klang-effekte.

n Ein ganzzahliger Wert zwischen 0 und 15, der die Lautstärke bestimmt. 0 bedeutet Ausschalten der Ton– oder Klang–Ausgabe.

Bemerkungen: Die VOL–Anweisung ist für alle wählbaren Stimmen (s. SOUND– und PLAY–Anweisungen) gültig.

WIDTH–Anweisung

Format: **WIDTH *n***

Zweck: Setzt die Strichstärke für alle grafischen Zeichenanweisungen (BOX, CIRCLE, DRAW etc.).

n Ein ganzzahliger Wert von 1 für einfache oder 2 für doppelte Strichstärke.

WINDOW–Anweisung

Format: **WINDOW** ***xlo,ylo,xru,yru***[,***lö***]

Zweck: Setzt ein Bildschirmfenster für Textausgabe und löscht dieses wahlweise.

xlo,ylo Spalten– und Zeilennummer der linken oberen Ecke des Textfensters.

xru,yru Spalten– und Zeilennummer der rechten unteren Ecke des Textfensters.

lö Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das Bildfenster nach der Definition gelöscht werden soll (1) oder nicht (0). Voreingestellt ist 0.

XOR–Funktion

Format: *v*=XOR(*n1*,*n2*)

Zweck: Liefert einen ganzzahligen Wert im Bereich zwischen 0 und 65535, der sich aus der Exclusiv–ODER–Verknüfung von zwei ganzen, vorzeichenlosen 16–Bit–Werten ergibt.

n1*,*n2 Zwei ganzzahlige Werte im Bereich zwischen 0 und 65535, die intern in 16–Bit–Zahlen umgewandelt werden, ehe sie logisch miteinander verknüpft werden.

Beispiele:

```
PRINT XOR(124,12);XOR(0,12)
112 12
READY.
```

4.7 Farben, Sprites und grafische Effekte

Der C128 besitzt 14 neue Befehle, die es Ihnen leicht machen, Grafiken zu erzeugen.

BOX	CHAR
CIRCLE	DRAW
COLOR	LOCATE
GRAPHIC	PAINT
MOVSPR	GSHAPE
SCALE	SSHAPE
SPRSAV	SPRITE

Genauere Informationen entnehmen Sie bitte Kapitel 4.6.

Grafische Besonderheiten

Ein hervorstechendes Merkmal des C128 ist seine grafische Programmierbarkeit. Er kann 16 verschiedene Farben (siehe untenstehende Tabelle) erzeugen. Außerdem besitzt er noch 5 verschiedene grafische Modi und kann 8 verschiedene bewegliche Objekte, genannt Sprites, verwalten.

Folgende Farbcodes sind verfügbar:

1	schwarz	9	hellbraun
2	weiß	10	braun
3	rot	11	rosa
4	grün	12	dunkelgrau
5	violett	13	grau
6	dunkelgrün	14	hellgrün
7	blau	15	hellblau
8	gelb	16	hellgrau

Folgende grafische Modi sind wählbar:

0	Text mit 40 Zeichen/Zeile.
1	hochauflösende Grafik (320x200 Punkte).
2	hochauflösende Grafik, geteilter Bildschirm für Grafik und Text.
3	Mehrfarbengrafik (160x200 Punkte)
4	Mehrfarbengrafik, geteilter Bildschirm für Grafik und Text.
5	Text mit 80 Zeichen/Zeile.

Hochauflösende und Mehrfarben–Grafik

Wie bereits erwähnt, zählen zu den Besonderheiten des C128 zwei verschiedene grafische Betriebsmodis, nämlich der Betrieb mit hochauflösender Grafik und der Mehrfarben–Grafikbetrieb, die mit der GRAPHIC–Anweisung gewählt werden können.

Bei beiden Betriebsarten (wie auch beim Mischbetrieb mit geteiltem Bildschirm) wird ein Speicherbereich von insgesamt 9 bzw. 10 KBytes reserviert. Von diesem Bereich werden 8000 Bytes für den Grafik–Bildschirm belegt.

Da jedes Byte aus 8 Bit besteht, stehen der Grafik also insgesamt 64000 Bits zur Verfügung. Jedem dieser Bits ist ein Bildpunkt der Grafik zugeordnet (320x200 = 64000 Bildpunkte). Man spricht deshalb auch von einem Bitmapped–Speicher.

Jedes Bit kann zwei Zustände annehmen, nämlich 0 (Null) und 1. 0 bedeutet aus und 1 bedeutet ein.

Der Baustein (VIC = Video Interface Controller), der nun alle Vorgänge auf dem Bildschirm sowie die Farbwiedergabe steuert, stellt unter anderem sowohl für den Hintergrund als auch für den Vordergrund je eine Farbe zur Verfügung, die aus einer Skala von 16 verschiedenen Farben wählbar ist.

Da bei der **hochauflösenden Grafik** jeder Bildpunkt durch ein Bit dargestellt wird, kann er auch nur zwei verschieden Farben annehmen, nämlich die Hintergrundfarbe, wenn das Bit 0 (aus) ist oder die Vordergrundfarbe, wenn das Bit 1 (ein) ist. Es sind also insgesamt nur 2 Farben bei hoher Auflösung möglich.

Bei der **Mehrfarbengrafik** wird die Auflösung auf 160x200 = 32000 Bildpunkte reduziert. Damit stehen im 64000–Bildpunkte–Speicher für jeden Bildpunkt 2 Bits zur Verfügung, die insgesamt vier verschiedene Zustände erlauben, nämlich:

00 01 10 11

Damit kann jetzt ein Bildpunkt neben der Hintergrund– und der Vordergrundfarbe zwei weitere Farben (Zusatzfarben 1 und 2) annehmen. Es sind bei der Mehrfarbengrafik also insgesamt 4 Farben möglich.

Welche Farben verwendet werden, wird mit der COLOR–Anweisung, deren Wirkung im folgenden noch näher erläutert wird, festgelegt. Mit ihr werden jeder der vier im Grafikbetrieb existierenden Farbquellen (Hintergrund, Vordergrund, Zusatzfarben 1 und 2) eine der 16 möglichen Farben zugewiesen. Diese Farbquellen und damit die ihnen zugeordneten Farben sind wiederum Parameter der grafischen Zeichenanweisungen BOX, CHAR, CIRCLE, DRAW und PAINT.

Erstellen von Grafik–Programmen im C128–Modus

Mit den oben beschriebenen Grundlagen sollte es nicht mehr allzu schwierig sein, eigene Grafik–Programme für den C128 zu erstellen.

Wenn Sie diesem Abschnitt schrittweise folgen, können Sie am Schluß selbst Grafik–Programme schreiben.

1. Zunächst müssen Sie die Farben für den Vorder– und Hintergrund sowie den Bildrahmen wählen. Dazu dient folgender Befehl:

COLOR *Farbquelle,Farbcode*

Die Farbquelle beschreibt den Teil des Bildschirms, den Sie farblich verändern wollen. *Farbcode* ist die Zahl, mit der Sie die Farbe bei der gewählten Farbquelle bestimmen (s. Tabellen unten):

Code	Quelle
0	Hintergund (40–Zeichen–Anzeige)
1	Vordergrund
2	Mehrfarbenmodus 1
3	Mehrfarbenmodus 2
4	Rand
5	Textfarbe
6	Hintergrund (80–Zeichen–Anzeige)

Code	Farbe	Code	Farbe
1	schwarz	9	hellbraun
2	weiß	10	braun
3	rot	11	rosa
4	grün	12	dunkelgrau
5	violett	13	grau
6	dunkelgrün	14	hellgrün
7	blau	15	hellblau
8	gelb	16	hellgrau

Für ein Beispiel mit dem Hintergrund schwarz, dem Vordergrund rot und dem Bildrahmen schwarz probieren Sie folgendes Programm aus.

```
10 COLOR 0,1
20 COLOR 1,3
30 COLOR 4,1
```

2. Der nächste Schritt bestimmt den Grafikmodus.

GRAPHIC ***Modus***[,[***Lösch***],[***Textzeile***]]

Modus muß eine Zahl zwischen 0 und 5 sein. Die unten stehende Tabelle zeigt, welche Zahlen dem jeweiligen graphischen Modus entsprechen.

Lösch ist ein ganzzahliger Wert von 0 oder 1, der bei Aufruf der Grafikmodi 1–4 bestimmt, ob der Bildschirm gelöscht werden soll (1) oder nicht (0).

Textzeile ist ein ganzzahliger Wert zwischen 0 und 24 (gilt nur für die Modi 2 und 4), der die Zeile festlegt, bei der der Text–Teilbildschirm beginnen soll. Voreingestellt ist 19.

Modus	Bedeutung
0	Text 40 Zeichen/Zeile
1	hochauflösende Grafik (320x200 Punkte)
2	hochauflösende Grafik (geteilter Bildschirm für Grafik und Text)
3	Mehrfarbengrafik (160x200 Punkte)
4	Mehrfarbengrafik (geteilter Bildschirm für Grafik und Text)
5	Text 80 Zeichen/Zeile

Das folgende Beispiel arbeitet mit der hochauflösenden Grafik. Geben Sie folgende Programmzeile ein:

```
40 GRAPHIC 1,1
```

Die zweite Zahl 1 im GRAPHIC–Befehl löscht den Bildschirm. Soll der Bildschirm nicht gelöscht werden, geben Sie an der zweiten Stelle eine 0 ein.

> **Beachte:**
> Arbeiten Sie nur mit einem Bildschirm und sollten Sie z.B. durch einen SYNTAX ERROR in der hochauflösenden Grafik "hängen" bleiben, drücken Sie die Funktionstaste f1 (GRAPHIC), dann 0 und dann RETURN. Dies bewirkt eine automatische Rückkehr zum Textmodus, wenn Sie mit dem 40–Zeichen–Bildschirm arbeiten und einen Grafik-Bildschirm aktiviert haben.

3. Nun können Sie damit beginnen, Ihre grafischen Befehle einzugeben.

Zunächst müssen Sie den grafischen Cursor (Pixel Cursor) an die Stelle des Bildschirms setzen, an der Sie zu zeichnen beginnen wollen. Dazu dient die LOCATE–Anweisung.

Geben Sie also folgende Programmzeile ein.

```
50 LOCATE 150,130
```

Diese Anweisung positioniert den grafischen Cursor auf dem Bildschirm (im Gegensatz zum Text–Cursor ist der grafische Cursor nicht sichtbar).

150 ist die x–Koordinate (horizontal) und 130 die y–Koordinate (vertikal) auf dem Bildschirm. Die x–Koordinate reicht von 0 – 319 und die y–Koordinate von 0 – 199 (in Bildpunkten).

Die aktuelle Position des grafischen Cursors können Sie mit der RDOT-Funktion (s. dort in Kapitel 4.6) ermitteln.

4. Jetzt lassen Sie uns einen Kreis zeichnen.

Geben Sie folgende Programmzeile ein:

```
60 CIRCLE 1,150,130,40,40
```

Die erste Zahl (1) bedeutet, daß der Kreis in der Vordergrundfarbe gezeichnet wird. Eine 0 bewirkt hier, daß der Kreis in der Hintergrundfarbe gezeichnet, also gelöscht wird.

Der Befehl bewirkt weiter, daß ein Kreis in der Mitte des Bildschirmes gezeichnet wird. Die CIRCLE–Anweisung kennt 9 verschiedene Parameter, damit können Sie die verschiedensten Kreise und geometrischen Figuren zeichnen.

Was die einzelnen Zahlen bedeuten, entnehmen Sie bitte der Beschreibung dieses Befehls im Kapitel 4.6.

5. Nun versuchen Sie, ein Rechteck zu zeichnen.

Geben Sie ein:

```
80 BOX 1,20,100,80,160,,1
```

Diese Anweisung bewirkt, daß links vom Kreis ein Quadrat gezeichnet wird. Die BOX–Anweisung (s. dort in Kapitel 4.6) besitzt sieben Parameter, die es Ihnen erlauben, verschiedene Rechtecke zu zeichnen.

Der erste Wert kann 0 oder 1 sein. Bei 1 wird das Quadrat in der Vordergrundfarbe gezeichnet, bei 0 in der Hintergrundfarbe, also gelöscht.
Der 2. und 3. Wert sind die x– und y– Koordinaten der oberen linken Ecke.
Der 4. und 5. Wert sind die x– und y– Koordinaten der unteren rechten Ecke.
Der letzte Wert kann 0 oder 1 sein. 0 bedeutet, daß nur die Umrahmung gezeichnet wird. 1 füllt das Rechteck aus.

Nun wechseln Sie die Farbe im Vordergrund und zeichnen ein weiteres Quadrat mit den unten stehenden Koordinaten.

Schreiben Sie also folgende Programmzeilen:

```
90 COLOR 1,9:REM FARBWECHSEL IM VORDERGRUND
100 BOX 1,220,100,280,160
```

6. Mit Hilfe der DRAW–Anweisung (s. dort in Kapitel 4.6) lassen sich auch einzelne Linien auf den Bildschirm zeichnen. Folgende Anweisung zieht eine Linie unterhalb der vorher gezeichneten Figuren:

```
120 DRAW 1,20,180 TO 280,180
```

Nun kurz erläutert, was die einzelnen Zahlen bedeuten. Der erste Wert 1 bedeutet Vordergrundfarbe. 20 und 180 sind die x– und y–Koordinaten für den Startpunkt. Die Werte 280 und 180 sind die x– und y–Koordinaten des Endpunkts. Probieren Sie einige DRAW–Anweisungen einmal selbst aus.

Die DRAW–Anweisung läßt sich auch so variieren, daß die gezeichneten Linien ihre Richtung ändern.

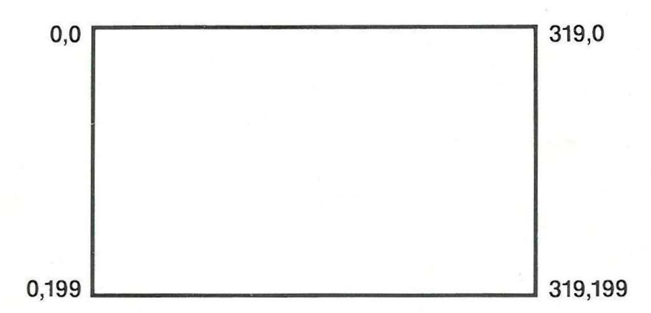
Versuchen Sie einmal folgende Programmzeile:

```
130 DRAW 1,10,20 TO 300,20 TO 150,80 TO 10,20
```

Sie sehen, daß der Computer nun ein Dreieck zeichnet. Die vier Wertepaare stehen für die jeweiligen Koordinaten der drei Seiten. Es ist also möglich, verschiedene mathematische Figuren zu zeichnen.

Es ist Ihnen vielleicht aufgefallen, daß sich der grafische Cursor, wenn x = 0 und y = 0 sind, am oberen linken Ende des Computers befindet (HOME–Position).

Die Abbildung zeigt Ihnen die Anordnung der x– und y–Koordinaten auf dem Bildschirm:



Es gibt auch die Möglichkeit, einzelne Punkte auf Ihrem Bildschirm zu zeichnen.

Geben Sie folgende Programmzeile ein:

```
150 DRAW 1,150,190
```

Mit dieser Anweisung wird ein Punkt unter dem Kreis gezeichnet.

7. *Mit der DRAW–Anweisung können Sie auch Bereiche des Bildschirms durch Umrandung hervorheben.*

Sollten Sie jedoch einmal eine Fläche mit Farbe ausmalen wollen, so benutzen Sie die PAINT–Anweisung (s. dort in Kapitel 4.6).

Geben Sie z.B. ein:

```
160 PAINT 1,150,97
```

Dieser Befehl malt den Kreis, den Sie in Programmzeile 60 gezeichnet haben, vollständig aus.

Nun zwei Befehle, die das oben gezeichnete Dreieck und das Quadrat rechts des Kreises ausmalen:

```
180 PAINT 1,50,25
200 PAINT 1,225,125
```

Hinweis:
Sie müssen bei der PAINT–Anweisung den Startpunkt innerhalb der auszumalenden Figur wählen, andernfalls wird die Figur nicht ausgemalt.

8. *Es ist auch möglich, Text im Grafik–Modus auf dem Bildschirm anzuzeigen.*

Geben Sie z.B. folgendes ein:

```
210 CHAR 1,10,1,"GRAFISCHES BEISPIEL"
```

Bei diesem Befehl wird der Text ab Spalte 10 in Zeile 1 angezeigt.

9. *Der Maßstab, in der eine grafische Aweisung auf dem Bildschirm zeichnet, kann mit der SCALE–Anweisung verändert werden.*

Damit können sie mehrere Figuren gleichzeitig auf dem Bildschirm abbilden. Im folgenden soll eine weitere Möglichkeit der SCALE–Anweisung erläutert werden.

Bei der hochauflösenden Grafik sind 320x200 Punkte möglich, im Mehrfarbenmodus jedoch nur 160x200. Der Nachteil der geringeren Auflösung wird durch 2 zusätzliche Farben gemildert. Sie haben also für jeden Bildpunkt jetzt 4 Farben zur Verfügung.

Koordinaten–Skalierung

Die SCALE–Anweisung dehnt den grafische Koordinatenbereich in x– und y–Richtung auf 1024 aus, unabhängig davon, ob in der hohen Auflösung oder im Mehrfarbenmodus gearbeitet wird.

SCALE 1 ändert den Koordinatenmaßstab (x/y von 0 bis 1023)

SCALE 0 stellt den Koordinatenmaßstab auf die voreingestellten Werte (x/y von 0 bis 319/199) zurück.

Die bisher beschriebenen und eingegebenen Befehlszeilen bilden ein Programm. Um jeder durch dieses Programm erstellten Figur eine andere Farbe zù geben, werden noch die Zeilen 70, 110, 140, 170, 190 und 210 mit COLOR–Anweisungen eingefügt (s. Programmliste weiter unten).

Geben Sie das Programm ein und starten Sie es durch Eingabe des RUN–Befehls (RETURN–Taste nicht vergessen).

Nachdem Sie alle Beispiele einmal ausprobiert haben, empfiehlt es sich, diese auf einer Diskette oder Kassette speichern zu, damit sie Sie später wieder verwenden können. Weitere Informationen über den Gebrauch der beschriebenen BASIC–Anweisungen finden Sie im Kapitel 4.6.

Programmbeispiel:

```
10 COLOR 0,1
20 COLOR 1,3
30 COLOR 4,1
40 GRAPHIIC 1,1
50 LOCATE 150,130
60 CIRCLE 1,150,130,40
70 COLOR 1,7: REM FARBE BLAU 1. QUADRAT
80 BOX 1,20,100,80,160,,1
90 COLOR 1,9 : REM RAHMENFARBE HELLBRAUN 2. QUADRAT
100 BOX ,220,100,280,160
110 COLOR 1,10: REM FARBE BRAUN FUER LINIEN
120 DRAW 1,20,180 TO 280,180
130 DRAW 1,10,20 TO 300,20 TO 150,80 TO 10,20
140 COLOR 1,6: REM FARBE GRUEN FUER PUNKT
150 DRAW 1,150,190: REM ZEICHNET PUNKT UNTERM KREIS
160 PAINT 1,150,97: REM FUELLT DEN KREIS AUS
170 COLOR 1,3: REM FARBE ROT FUER DAS DREIECK
180 PAINT 1,50,25: REM FUELLT DAS DREIECK AUS
190 COLOR 1,2: REM FARBE WEISS FUER DAS 2. QUADRAT
200 PAINT 1,225,125: REM FUELLT DAS 2. QUADRAT AUS
210 COLOR 1,14: REM SCHREIBT TEXT HELLGRAU
220 CHAR 1,10,1,"GRAFISCHES BEISPIEL"
```

Sprites: Programmierbare und bewegbare Objekte

Ihr C128 bietet Ihnen ferner die Möglichkeit, selbst entworfene Figuren auf dem Bildschirm zu bewegen. Dieses geschieht mit der SPRITE–Anweisung. Sie brauchen zum Erstellen von Sprites kein Millimeterpapier mehr, wenn sie mit dem eingebauten Sprite–Editor (s. a. SPRDEF–Befehl im Kapitel 4.6) arbeiten.

Im folgenden wird beschrieben, wie Sie Sprites entwerfen und benutzen können.

Sollten Sie schon einmal mit dem C64 gearbeitet haben, so kennen Sie schon die Grundbegriffe der Sprite–Grafik. Die Sprite–Grafik befähigt den Computer, Figuren jeglicher Form beliebig über den Bildschirm laufen zu lassen. Sprites lassen sich in 16 verschiedenen Farben zeichnen und jedes Sprite kann bunt dargestellt werden. Die folgenden Anweisungen erlauben Entwurf und Benutzung von Sprites.

SPRDEF
SSHAPE
GSHAPE
SPRSAV
SPRITE
MOVSPR
SPRCOLOR
COLLISION

Sprite–Entwurf

Als erstes müssen Sie sich über das Aussehen des von Ihnen gewünschten Sprites im klaren sein, bevor Sie es auf dem Bildschirm bewegen können (z.B. eine Rakete oder einen Rennwagen).

Im C128–Modus können Sie auf 3 verschiedene Arten Sprites erzeugen:

1. Mit den SSHAPE–, SPRSAV– und SPRITE–Anweisungen.
2. Mit dem SPRDEF–Befehl.
3. Mit POKE–Anweisungen wie beim C64 (s. Kap. 5.3.1).

1. Der Gebrauch der SSHAPE–, SPRSAV– und SPRITE–Anweisungen

Die Verwendung dieser Anweisungen erlaubt auf schnelle und einfache Art, Sprites als Teile einer Grafik zu erzeugen. Die einzelnen Anweisungen sind in Kapitel 4.6 dieses Handbuches beschrieben.

Hier die Beschreibung eines Sprite–Entwurfsvorgangs:

- Zeichnen Sie ein beliebiges Bild mit Hilfe der DRAW–, CIRCLE–, BOX– und PAINT–Anweisungen aus dem vorherigen Kapitel. Die äußeren Grenzen dieser Zeichnung werden durch eine 24 x 21–Punktmatrix im hochauflösenden Modus oder eine 12 x 21–Punktmatrix im Mehrfarbenmodus festgelegt.

- Benutzen Sie die SSHAPE–Anweisung, um dieses Bild in einer Zeichenketten–Variablen zu speichern.

- Übertragen Sie Ihren Entwurf mit Hilfe der SPRSAV–Anweisung in die Sprite–Grafik.

- Aktivieren Sie das Sprite mit der SPRITE–Anweisung, malen Sie es aus, wählen Sie einen Farbmodus und vergrößern Sie das Sprite.

- Bewegen Sie das Sprite mit Hilfe der MOVSPR–Anweisung.

2. Der SPRDEF–Befehl

Der SPRDEF–Befehl ist der schnellste und einfachste Weg, Sprites zu erzeugen. Der SPRDEF–Befehl schaltet den Sprite–Editor ein. Genauere Informationen entnehmen Sie bitte der Befehlsbeschreibung im Kapitel 4.6.

Im folgenden sehen Sie an einem detaillierten Beispiel, wie Sie ein Sprite als Teil einer Grafik selbst entwerfen können.

Dieses Beispiel arbeitet mit der hochauflösenden Grafik. Für Umschaltung in den Grafikmodus bei schwarzem Hintergrund geben Sie folgende Programmzeilen ein:

```
5 COLOR 0,1:REM HINTERGRUND SCHWARZ
10 GRAPHIC 1,1:REM HOCHAUFLOESENDE GRAFIK
```

Das folgende kurze Programm zeichnet einen Rennwagen am oberen linken Bildrand:

```
15 BOX 1,2,2,45,45:REM ZEICHNET RECHTECK UMS AUTO
20 DRAW 1,17,10 TO 28,10 TO 26,30 TO 19,30 TO
        17,10:REM KAROSSERIE
22 DRAW 1,11,10 TO 15,10 TO 15,18 TO 11,18 TO
        11,10:REM OBERES LINKES RAD
24 DRAW 1,30,10 TO 34,10 TO 34,18 TO 30,18 TO
        30,10:REM OBERES RECHTES RAD
26 DRAW 1,11,20 TO 15,20 TO 15,28 TO 11,28 TO
        11,20:REM UNTERES LINKES RAD
28 DRAW 1,30,20 TO 34,20 TO 34,28 TO 30,28 TO
        30,20:REM UNTERES RECHTES RAD
30 DRAW 1,26,28 TO 19,28:REM KUEHLERGRILL
32 BOX 1,20,14,26,18,90,1:REM WINDSCHUTZSCHEIBE
35 BOX 1,150,35,195,40,90,1:REM STRASSE
37 BOX 1,150,135,195,140,90,1:REM STRASSE
40 BOX 1,150,215,195,220,90,1:REM STRASSE
42 DRAW 1,50,180 TO 300,180:DRAW 1,50,180 TO
        50,190:REM ZIELLINIE
43 DRAW 1,300,180 TO 300,190:DRAW 1,50,190 TO
        300,190:REM ZIELLINIE
44 CHAR 1,18,23,"Z I E L":REM SCHREIBT ZIEL
```

Lassen Sie das geschriebene Programm einmal laufen. Sie sehen in der oberen rechten Ecke einen weißen Rennwagen in einem Rechteck. Außerdem haben Sie eine angedeutete Rennstrecke in dem unteren Feld mit einer Ziellinie gezeichnet. Das Auto ist noch kein Sprite, denn es fehlt ihm noch die Fähigkeit, sich zu bewegen. Dazu ist es notwendig, das gerade gezeichnete Bild in einer Zeichenkettenvariablen zu speichern.

Fügen Sie hierzu folgende Zeile in ihr Programm ein:

```
45 SSHAPE A$,11,10,34,31:REM SPEICHERN DES BILDES IN
        EINEM STRING
```

Die Zahlen 11,10,34,31 sind die Koordinaten der linken oberen und rechten unteren Ecke des Bildes. Sie müssen die Position der Koordinaten genau bestimmen, sonst besteht die Gefahr, daß das Sprite nicht korrekt oder gar nicht gespeichert wird.

Die Variable **A$** speichert eine binäre Zeichenkette, die aus Nullen und Einsen besteht und im Speicher des Computers abgelegt wird. Jedem Punkt des Bildschirms ist ein Bit im grafischen Speicher des Computers zugeordnet. Ist das Bit an (1), ist der Punkt in der gewählten Vordergrundfarbe sichtbar. Ist das Bit aus (0), hat der Punkt die gewählte Hintergrundfarbe.

Ihr Bild ist nun gespeichert. Im nächsten Schritt wird das in **A$** gespeicherte Bild in ein Sprite übertragen.

Dies kann durch folgendene Anweisungen realisiert werden:

SPRSAV ***n,v$***
SPRSAV ***v$,n***

Mit dieser Anweisung können Sie die in einer Zeichenkettenvariablen abgelegten Bitmuster als Sprite–Daten verwenden bzw. das Bitmuster eines Sprites an eine Zeichenkettenvariable übergeben.

n Ein ganzzahliger Wert zwischen 1 und 8 für die Sprite–Nummer.
v$ Eine beliebige Zeichenkettenvariable, die die Binärinformation aufnimmt oder enthält.

Die folgenden SPRSAV–Anweisungen legen das in **A$** gespeicherte Bild in zwei Sprites ab.

Machen Sie dazu folgende Ergänzung zu dem bisherigen Programm:

```
50 SPRSAV A$,1:REM SPEICHERT INHALT VON A$ IN SPRITE 1

55 SPRSAV A$,2:REM SPEICHERT INHALT VON A$ IN SPRITE 2
```

Bevor Sie die Sprites nun sehen können, müssen Sie diese erst aktivieren.

Dies geschieht durch die SPRITE–Anweisung, die auch die Größe des Objekts bestimmen kann. Außerdem werden der grafische Modus entweder als hochauflösend oder mehrfarbig durch die SPRITE–Anweisung festgelegt sowie die Vorder– und Hintergrundfarben definiert
Die folgenden Programmzeilen aktivieren die Sprites:

```
60 SPRITE 1,1,7,0,0,0,0:REM AKTIVIERT SPRITE 1

65 SPRITE 2,1,3,0,0,0,0:REM AKTIVIERT SPRITE 2
```

Im folgenden werden die einzelnen Parameter der SPRITE–Anweisung erklärt.

SPRITE ***n***[,[***akt***][,***Farbe***][,***Pri***][,***xdehn***]
[,[***ydehn***][,[***Modus***]]]]]]

n – Ganzzahliger Wert zwischen 1 und 8 für die Sprite–Nummer.
akt – Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das Sprite aktiviert (1) oder inaktiviert (0) ist.
Farbe – Ganzzahliger Wert zwischen 1 und 16 für die Sprite–Farbe. Die Zuordnung der Codes zu den Farben sind in der COLOR–Anweisung (Kapitel 4.6) beschrieben.

Pri – Ein ganzzahliger Wert von 0 oder 1. Bei 0 wandert das Sprite während der Bewegung vor ggfs. vorhandenen Bildschirmobjekten vorbei, bei 1 dahinter.

xdehn – Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das Sprite in x–Richtung gedehnt ist (1) oder nicht (0).

ydehn – Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das Sprite in y–Richtung gedehnt ist (1) oder nicht (0).

Modus – Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das Sprite im hochauflösenden (0) oder im Mehrfarben–Modus (1) angezeigt werden soll.

Die MOVSPR–Anweisung gibt Ihrem Sprite nun die Bewegung. Sie plaziert oder bewegt ein Sprite an eine bestimmte Stelle des Bildschirms. Ihre allgemeine Form ist:

MOVSPR ***n,x,y***
MOVSPR ***n,+/–x1,+/–y1***
MOVSPR ***n,Winkel#Geschw***

n Ein ganzzahliger Wert zwischen 1 und 8, der die Nummer des Sprites angibt.

x,y Die absoluten Koordinaten, bei denen das Sprite positioniert werden soll.

+/–x1,+/–y1 Die zur letzten Sprite–Position relativen Koordinaten, bei denen das Sprite positioniert werden soll.

Winkel Ein ganzzahliger Wert zwischen 0 und 360, der de r angibt, unter welchem Winkel das Sprite bewegt werden soll.

Fügen Sie bitte folgende Zeilen an Ihr Progamm an:

```
70 MOVSPR 1,240,0:REM POSITION DES SPRITES 1

80 MOVSPR 2,120,0:REM POSITION DES SPRITES 2
```

Zeile 70 bewirkt, daß das Sprite 1 zu der Position 240/0 des Punktkoordinatensystems bewegt wird.
Zeile 80 bewirkt analog, daß das Sprite 2 zu der Position 120/0 des Punktkoordinatensystems bewegt wird.

Folgende Programmzeilen bewegen die Sprites entlang einer definierten Linie:

```
85 MOVSPR 1,180#6:REM BEWEGT SPRITE 1 VON OBEN
                      NACH UNTEN
87 MOVSPR 2,180#7:REM BEWEGT SPRITE 2 VON OBEN
                      NACH UNTEN
```

Der erste Wert dieser Anweisung ist die Sprite–Nummer. Der zweite Wert (180) ist der Winkel, unter dem das Sprite vom Ausgangspunkt wegbewegt wird. Die letzte Zahl gibt die Geschwindigkeit an, mit der das Sprite bewegt wird.

Nun lassen Sie einmal Ihr Programm durchlaufen und Sie sehen, daß es Ihnen gelungen ist, eine Rennbahn mit zwei Rennwagen zu programmieren. Lassen Sie jetzt Ihrer Phantasie freien Lauf und probieren Sie eigene Objekte aus.

Es folgt noch einmal das komplette Programm:

```
5 COLOR 0,1
10 GRAPHIC 1,1
15 BOX 1,2,2,45,45
20 DRAW 1,17,10 TO 28,10 TO 26,30 TO 19,30 TO 17,10
22 DRAW 1,11,10 TO 15,10 TO 15,18 TO 11,18 TO 11,10
24 DRAW 1,30,10 TO 34,10 TO 34,18 TO 30,18 TO 30,10
26 DRAW 1,11,20 TO 15,20 TO 15,28 TO 11,28 TO 11,20
28 DRAW 1,30,20 TO 34,20 TO 34,28 TO 30,28 TO 30,20
30 DRAW 1,26,28 TO 19,28
32 BOX 1,20,14,26,18,90,1
35 BOX 1,150,35,195,40,90,1
37 BOX 1,150,135,195,140,90,1
40 BOX 1,150,215,195,220,90,1
42 DRAW 1,50,180 TO 300,180:DRAW 1,50,180 TO 50,190
43 DRAW 1,300,180 TO 300,190:DRAW 1,50,190 TO 300,190
44 CHAR 1,18,23,"ZIEL"
45 SSHAPE A$,10,10,33,31
50 SPRSAV A$,1
55 SPRSAV A$,2
60 SPRITE 1,1,7,0,0,0,0
65 SPRITE 2,1,3,0,0,0,0
70 MOVSPR 1,240,0
80 MOVSPR 2,120,0
85 MOVSPR 1,180 # 6
87 MOVSPR 2,180 # 7
```

Zeile 5 –	Der Hintergrund des Bildschirm wird schwarz.
Zeile 10 –	Schaltet die hochauflösenden Grafik ein und löscht den Bildschirm.
Zeile 15 –	Zeichnet ein Rechteck oben links.
Zeilen 20–32 –	Zeichnen ein Rennauto.
Zeilen 35–44 –	Zeichnen die Rennbahn und die Ziellinie.
Zeilen 50–55 –	Übertragen den Teil der Zeichenketten–Variablen in die Sprites 1 und 2.
Zeilen 70–80 –	Bewegen die Sprites zum oberen Bildschirmrand.
Zeilen 85–87 –	Bewegen die Sprites von oben nach unten.

Verbinden von Sprites zu einem gößeren Objekt

Wollen Sie größere Figuren auf Ihrem Bildschirm erstellen, müssen Sie einzelne Sprites zu der gewünschten Gesamtfigur zusammenfügen. Die folgende Anleitung zeigt Ihnen, wie Sie zwei Sprites zusammenfügen können:

- Malen Sie wieder ein beliebiges Bild mit Hilfe der DRAW–, BOX– und PAINT–Anweisungen. Verdoppeln Sie diesmal jedoch die Größe Ihrer Fläche auf 48 x 21 Punkte.
- Benutzen Sie zwei SSHAPE–Anweisungen, um die zwei Sprites einzeln zu speichern. Teilen Sie nun ihre 48 x 21 Punkte große Bildfläche in zwei gleichgroße Bildflächen (24 x 21) und speichern Sie die Sprites mit Ihren entsprechenden Koordinaten. Benutzen Sie zwei verschiedene Zeichenkettenvariablen (A$).
- Übertragen Sie die Daten des Bildes in separate Sprites mit Hilfe des SPRSAV–Befehls.
- Aktivieren Sie die Sprites mit der SPRITE–Anweisung.
- Positionieren Sie das zweite Sprite so, daß sich der erste Punkt des zweiten Sprites mit dem letzten Punkt des ersten deckt. Jetzt haben Sie die Sprites miteinander verbunden.

Probieren Sie nun noch folgende Programmzeilen aus:

```
100 MOVSPR 1,10,10
```

Der erste Wert ist die Sprite–Nummer.
Diese Anweisung plaziert das erste Sprite an die Position x=10/y=10. Nun positionieren Sie das zweite Sprite 24 Punkte rechts vom ersten:

```
200 MOVSPR 2,34,10
```

- Jetzt ist es Ihnen möglich, Sprites auf dem Bildschirm zu einer von Ihnen gewünschte Position zu bewegen.

Zum Abschluß noch ein Beispiel für verbundene Sprites:

```
10 COLOR 0,1 : REM HINTERGRUND SCHWARZ
20 GRAPHIC 1,1 : REM LOESCHT BILDSCHIRM
30 BOX 1,1,1,51,25 : REM RECHTECK UM SPRITES
40 DRAW 1,3,3 TO 40,3 TO47,12 TO 0,19 TO 3,3
45 REM ZEICHNET EINE RAKETE
50 CHAR 1,1,1"RAKT" : REM SCHREIBT RAKT IN DIE RAKETE
60 SSHAPE C$,2,2,25,22 : REM SICHERT TEIL 1 IN C$
70 SSHAPE D$,26,2,48,22 : REM SICHERT TEIL 2 IN D$
80 SPRSAV C$,1 : REM SPEICHERT C$ IN SPRITE 1
90 SPRSAV D$,2 : REM SPEICHERT D$ IN SPRITE 2
100 SPRITE 1,1,2,0,0,0,0 : REM AKTIVIERT SPRITE 1
110 SPRITE 2,1,2,0,0,0,0 : REM AKTIVIERT SPRITE 2
120 MOVSPR 1,80,100 : REM VERSCHIEBT SPRITE 1
130 MOVSPR 2,104,100 : REM VERSCHIEBT SPRITE 2
```

4.8 Klänge und Musik mit dem C128

Der C128 hat im Vergleich zu anderen Microcomputern einen äußerst leistungsfähigen Musik- und Geräusch-Synthesizer. Der Klangbaustein (SID-Chip) befähigt den C128, drei verschiedene Töne gleichzeitig zu erzeugen. Jeder Ton kann außerdem in vier verschiedenen Klangfarben gespielt werden. Der Synthesizer erlaubt programmierbare Anschlagzeit, Abschwellzeit, Haltezeit und Ausklingzeit (ADSR- Parameter) und hat die Fähigkeit, bestimmte Töne hervorzuheben oder auszublenden. Außerdem besitzt der SID ein programmierbares Filter, mit dem Klänge und Geräusche verändert werden können.

Um die Möglichkeiten des SID-Bausteins voll zu nutzen, bietet das BASIC 7.0 des C128 eine Reihe neuer Anweisungen. Sie machen das Programmieren von Musik schneller und einfacher.

Um Ihre eigenen "Videoclips" zu erstellen, können Sie jetzt Ihre Grafikprogramme mit eigener Musik kombinieren. Dies sind die neuen Musik-Anweisungen dafür:

SOUND
ENVELOPE
VOL
TEMPO
PLAY
FILTER

Wie in den vorherigen Abschnitten wird jeder einzelne Befehl kurz erklärt. Im Laufe dieser Beschreibungen entsteht dann wieder ein kleines Programm.

Die **SOUND**–Anweisung ermöglicht es Ihnen, schnell und einfach Ton– oder Klangeffekte wählbarer Frequenz und Dauer zu erzeugen und sie über den Monitor oder ein Fernsehgerät auszugeben:

SOUND ***Stimme,Frequenz,Dauer***[,[***Richtung***]
[,[***Maxfr***][,[***Stufe***][,[***Welle***][,***Impbr***]]]]]

Die einzelnen Parameter bedeuten:

Stimme – Sie können von 1 bis 3 wählen:

1 Stimme 1 (Ton)
2 Stimme 2 (Ton)
3 Stimme 3 (Ton)

Frequenz – Ein ganzzahliger Wert zwischen 0 und 65535 wählt die Frequenz.

Dauer – Sagt, wie lange der Ton anhalten soll (in Schritten von 60zigstel Sekunden). Ein ganzzahliger Wert zwischen 0 und 32767.

Richtung – ganzzahliger Wert zwischen 0 und 2.

Maxfr – ganzzahliger Wert zwischen 0 und 65535.

Stufe – ganzzahliger Wert zwischen 0 und 32767.

Welle – ganzzahliger Wert zwischen 0 und 3 mit folgender Bedeutung:

0 Dreieck
1 Sägezahn
2 Rechteck
3 Rauschen

Voreingestellt ist 2.

Impbr – ganzzahliger Wert zwischen 0 und 4095. Bestimmt das Tastverhältnis.

Nähere Angaben über die SOUND–Anweisung entnehmen Sie bitte dem Kapitel 4.6.

Wollen sie z.B. einen Ton von 1 Sekunde erzeugen, muß der Wert für die Dauer von einer Skunde 60, für 10 Sekunden 600 usw. sein.

Nun probieren Sie bitte folgendes kurzes Programm:

```
10 VOL 5
20 SOUND 1,512,60
```

Lassen Sie das Programm laufen. Der C128 spielt einen kurzen Ton. Die Zeile 10 bestimmt die Lautstärke. Die Zahl 512 in Zeile 20 bestimmt die Frequenz.

Ändern Sie jetzt die Tonhöhe mit folgender Zeile:

```
30 SOUND 1,1000,60
```

Sie hören, daß der Ton wesentlich höher ist als der vorhergehende. Allerdings werden beide Töne nur eine Sekunde gespielt.

Versuchen Sie folgendes:

```
40 SOUND 1,0,60
```

Sie hören den tiefsten Ton, der möglich ist.

Nun durchlaufen Sie die von 0–65535 reichenden Tonhöhen in einer Schleife (FOR . . . NEXT):

```
50 FOR I = 0 TO 65535
60 SOUND 1,I,1
70 NEXT
```

Dieser Programmteil durchläuft den gesamten definierbaren Frequenzbereich von unten nach oben.

Ersetzen Sie jetzt die Zeile 60 durch die folgende:

```
60 SOUND 3,I,1,,,,3
```

Jetzt durchläuft das Programm den gesamten Bereich des Rauschgenerators. Die niedrigen Frequenzen klingen wie Rumpeln, die hohen Frequenzen wie eine startende Rakete.

Probieren Sie einmal folgende Variante aus. Es handelt sich dabei um ein Programm, das eine Flugzeugschlacht durch verschiedene Geräusche simuliert.

```
10 REM FLUGZEUGSCHLACHT
20 VOL 5
30 FOR I = 850 TO 1015
40 SOUND 3,I,1
50 SOUND 1,1015-I,1
60 NEXT
70 SOUND 3,965,60,,,,3
80 FOR I = 1023 TO 850 STEP -1
90 SOUND 3,I,1
100 SOUND 1,I-150,1
110 NEXT
120 FOR J = 1000 TO 0 STEP -7
130 SOUND 3,J,1
140 NEXT
150 SOUND 3,965,60,,,,3
160 GOTO 30
```

Durch Drücken der STOP–Taste können Sie das Programm beenden.

Im folgenden werden noch einmal die einzelnen Befehlszeilen erläutert:

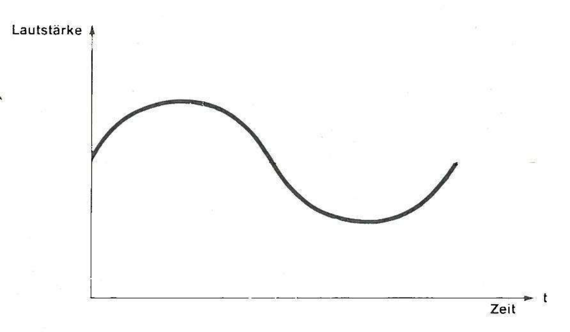
Zeile 20 stellt die Lautstärke auf 5.
Zeilen 30–60 bewirken das eigentliche Fluggeräusch. Wie Sie sehen und hören, nimmt die Tonhöhe bei Zeile 30 ab und bei Zeile 50 zu.
Zeile 70 erzeugt ein Maschinengewehrgeräusch.
Zeilen 80–110 produzieren das Geräusch eines abstürzenden Flugzeugs. Damit der Absturz realistischer wirkt, setzen die Tonhöhen unterschiedlich ein (s. Zeile 90 u. 100).
Zeilen 120–140 erzeugen das Detonationsgeräusch des Flugzeugs beim Aufprall.
Zeile 150 entspricht der Zeile 70.
Zeile 160 veranlaßt die Programmwiederholung ab Zeile 30.

Die SOUND–Anweisung ist eigentlich nur dazu gedacht, musikalische Nebeneffekte zu erzielen. Wollen Sie richtige Lieder auf ihrem C128 spielen, können Sie andere Befehle nutzen.

Geräusche und Musik im C128–Modus

Zu Beginn einige Grundlagen:

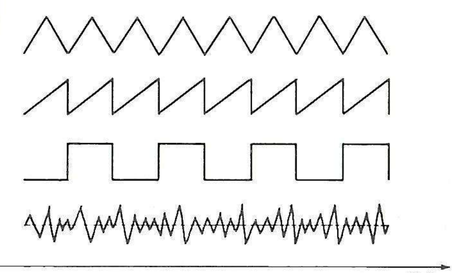
Jeder Ton, den sie hören, ist eine durch die Luft übertragene Welle. Diese Tonwelle bewegt sich mit einer bestimmten Geschwindigkeit (Frequenz).



Die Klangfarbe der Töne wird durch ihre Wellenform bestimmt.

Der C128 kann vier verschiedene Wellenformen erzeugen:

- Dreieck
- Zägezahn
- Rechteck
- Rauschen.



Musikerzeugung mit dem C128

Zum Verändern der Klangfarbe eines Tones können Sie die ENVELOPE–Anweisung benutzen. Sie kontrolliert die ADSR–Parameter und die Wellenform.

Das allgemeine Format der ENVELOPE–Anweisung ist wie folgt:

ENVELOPE *n*[,[*An*][,[*Ab*][,[*Ha*][,[*Au*][,[*Wf*]
[,[*Ib*]]]]]]

Die einzelnen Parameter bedeuten:

n – Nummer der Hüllkurve (0 bis 9).
An – Anschlagzeit (0 bis 15).
Ab – Abschwellzeit (0 bis 15).
Ha – Haltezeit (0 bis 15).
Au – Ausklingzeit (0 bis 15).
Wf – Wellenform :
0 Dreieck
1 Sägezahn
2 Rechteck
3 Rauschen
4 Ringmodulation

Ib – Impulsbreite (0 bis 4095).

Lassen Sie uns die einzelnen Begriffe wie Hüllkurve, Wellenform und Impulsbreite erläutern:

Die **Hüllkurve** ist das Zusammenspiel der einzelnen Faktoren (ADSR–Parameter), die den Lautstärkeverlauf eines Tones ausmachen (Anschlagzeit, Abschwellzeit, Haltezeit und Ausklingzeit). Eine Gitarre hat z.B. andere ADSR–Parameter als eine Flöte.

Unter der **Wellenform** versteht man die Art der Wellen, die z.B. ein Musikinstrument erzeugt. Sie entstehen durch die Kombination von musikalischen Harmonischen eines Tones. Harmonische sind das Vielfache des Grundtones.

Die **Impulsbreiten** sind Veränderungen der Rechteckform, wodurch sich auch das Verhältnis der Harmonischen zueinander verändert.

Sie sehen also, daß die ENVELOPE–Anweisung die Programmierung der grundlegeneden Musikbestandteile erlaubt.

Der C128 bietet bereits Klangfarben für 10 verschiedene Musikinstrumente, deren ADSR–Parameter bereits voreingestellt sind und die mit der ENVELOPE–Anweisung noch verändert werden können. Sie brauchen nur die entsprechende Hüllkurven–Nummer einzugeben und der C128 liefert die vorprogrammierten Parameter selbst.

Wenn Sie eigene Hüllkurven mit neuen Klangformen erstellen wollen, müssen Sie die einzelnen Parameter der ENVELOPE–Anweisung ändern.

Weitere Einzelheiten zu den vorprogrammierten Hüllkurven entnehmen Sie bitte der nachfolgenden Tabelle:

0	0	9	0	0	2	1536	Klavier
1	12	0	12	0	1		Akkordeon
2	0	0	15	0	0		Zirkusorgel
3	0	5	5	0	3		Trommel
4	9	4	4	0	0		Flöte
5	0	9	2	1	1		Gitarre
6	0	9	0	0	2	512	Cembalo
7	0	9	9	0	2	2048	Orgel
8	8	9	4	1	2	512	Trompete
9	0	9	0	0	0		Xylophon

Geben Sie nun die folgende Programmzeile ein.

```
10 ENVELOPE 9,10,5,10,5,2,4000:PLAY"T9"
```

Der nächste Schritt bestimmt die Lautstärke ihres Musikprogramms.

```
20 VOL 8
```

Die VOL–Anweisung gibt die Lautstärke zwischen 0 und 15 an, wobei 0 das Mininum (aus) und 15 das Maximum ist.

Mit der PLAY–Anweisung können Sie Noten spielen:

PLAY"*Zeichenkette*"

Zeichenkette – ist eine Zeichenkette aus speziellen Buchstabenbefehlen, die die Tonerzeugung und Abspielfolge festlegen. Die Bedeutung der einzelnen Parameter entnehmen Sie bitte dem Kapitel 4.6.

Der nächste Schritt kontrolliert das Spieltempo für die gewünschten Noten. Die TEMPO–Anweisung hilft Ihnen dabei.

Hier ist die allgemeine Form:

```
TEMPO n
```

n muß ein ganzzahliger Wert zwischen 0 und 255 sein. Fehlt in ihrem Programm dieser Befehl, so setzt der Computer die Länge 8 fest. Die tatsächliche Dauer berechnet sich:

Dauer = 19.22/***n*** in Sekunden.

Setzen Sie das Tempo = 0, so wird die Note gehalten.

Schreiben Sie nun folgende Programmzeile:

```
30 TEMPO 100
```

Es wird jetzt Zeit, zu lernen, wie Sie ein Lied spielen können. Dies geschieht analog zum PRINT–Befehl, indem Sie die gewünschten Noten bei der PLAY–Anweisung in Anführungzeichen eingeben.

Mit folgender PLAY–Anweisung spielt der Computer die Tonleiter :

```
40 PLAY "C D E F G A B"
```

Anm.: Der Note B entspricht im Deutschen die Note H.

Sie können nun folgende Zeichen vor die einzelnen Noten setzen.

– Die Note wird einen halben Ton höher gespielt.

$ – Die Note wird einen halben Ton niedriger gespielt.

. – verlängert die Note um die Hälfte ihres Wertes.

Sie können außerdem die Dauer der Töne variieren, indem Sie folgende Buchstaben vor die einzelnen Noten setzen:

W – Die folgende Note wird als ganze Note gespielt.

H – Die folgende Note wird als halbe Note gespielt.

Q – Die folgende Note wird als viertel Note gespielt.

I – Die folgende Note wird als achtel Note gespielt.

S – Die folgende Note wird als sechzehntel Note gespielt.

Durch ein in Anführungsstriche gesetztes "R" erreichen sie, daß der Ton angehalten wird.

Es ist weiter möglich, den Synthesizer des C128 durch zusätzliche Steuerzeichen in der PLAY–Anweisung für unterschiedliche Effekte beim Abspielen von Musik zu programmieren.

Diese folgen in einer Tabelle:

Kontrollzeichen	Beschreibung	Bereich	Voreinstell.
U*n*	Lautstärke	0 – 9	9
T*n*	Hüllkurve	0 – 9	0
V*n*	Stimme	1 – 3	0
X*n*	Filter	0 = an, 1 = aus	0
O*n*	Oktave	0 – 6	4

Normalerweise brauchen Sie die Steuerzeichen nicht zu benutzen. Sie können jedoch damit die Möglichkeiten des Synthesizers erheblich differenzierter ausschöpfen. Wenn Sie in der PLAY–Anweisung mit T*n* eine Hüllkurve definieren, so achten Sie darauf, daß der gewählte Wert die Hüllkurve betrifft, für die Sie mit der ENVELOPE–Anweisung Ihre spezifischen Parameter gesetzt haben.

Weil der SID–Chip nur ein Filter hat, wird es bei allen drei Stimmen genutzt.

Fügen Sie folgende Programmzeilen an Ihr bisheriges Programm und achten Sie auf den Unterschied zu Zeile 40:

```
50 PLAY"U5 V1 O5 C D E F G A B"
```

Diese Zeile spielt ebenfalls die Tonleiter, allerdings ist die Lautstärke auf 5 heruntergesetzt, die Stimme 1 wurde gewählt und die Noten werden eine Oktave (5) höher gespielt. Nun spielt Ihr Programm zwei paarallele Stimmen.

Eine 3. Stimme bei eingeschaltetem Filter können Sie durch die folgende Zeile erreichen:

```
60 PLAY "U7 V3 O6 X1 C D E F G A B"
```

Hier werden noch einmal die einzelnen Zeichen der Zeile 60 kurz erklärt.

U7 – setzt die Lautstärke auf 7
V3 – wählt die 3. Stimme
O6 – setzt die 3. Stimme auf die 6. Oktave
X1 – stellt das Filter an

Nähere Angaben entnehmen Sie bitte der Beschreibung der PLAY-Anweisung in Kapitel 4.6.

Ihr Programm spielt nun alle drei Stimmen. Jede eine Oktave höher als die vorhergehende.

Bis zu diesem Zeitpunkt haben Sie nur ganze Noten gespielt. In einem weiteren Programmbeispiel sollen auch halbe, achtel usw. Noten gespielt werden.

Ändern Sie die Noten, indem Sie die oben genannten Buchstaben einfügen:

```
70 PLAY "U7 V2 O6 X1 H CD Q EF I GA S B"
```

Zeile 70 spielt in der 6. Oktave mit der Stimme 2 in der Lautstärke 7. Das Filter ist eingeschaltet.

Hier eine Erläuterung der zusätzlichen Steuerzeichen in der PLAY- Anweisung:

C und D werden als halbe Noten gespielt.
E und F werden als viertel Noten gespielt.
G und A werden als achtel Noten gespielt.
B wird als sechzehntel Note gespielt.

Nun ergänzen Sie die Noten, indem Sie die Töne um eine halbe Oktave erhöhen, erniedrigen oder die Töne verlängern:

```
80 PLAY "U8 V0 O4 X0 .H CD Q# EF I$ GA .S# B"
```

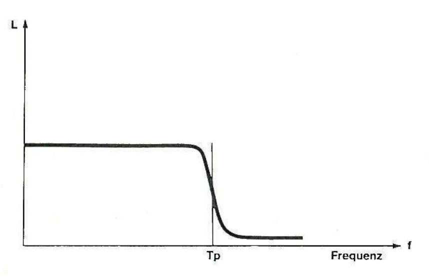
C und D werden als halbe Noten um die Hälfte ihres Wertes verlängert.
E und F – werden als viertel Noten eine halbe Oktave höher gespielt.
G und A – werden als achtel Noten eine halbe Oktave niedriger gespielt.
B – wird um die Hälfte des Wertes verlängert und einen halben Ton höher gespielt.

Wenn Sie schon die Wellenform, ADSR–Parameter, Lautstärke und Tempo gewählt haben, sollten Sie nun das Filter wählen. Die FILTER–Anweisung verändert ebenfalls Klangfarbe und –höhe.

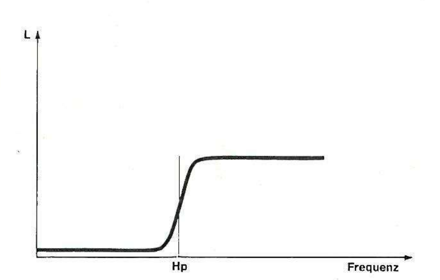
Das im SID eingebaute Filter gibt Ihnen die Möglichkeit die Wellenform hervorzuheben oder zu eleminieren. Damit ändert das Filter das Timbre (Klangfarbe) eines Tons.

Der SID–Chip kann drei verschiedene Filtertypen simulieren:

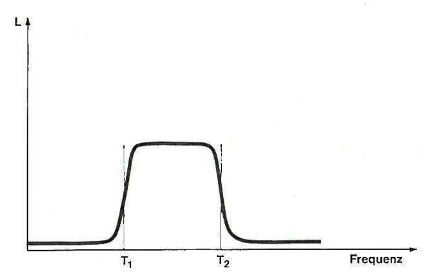
1. Tiefpass Filter
2. Hochpass Filter
3. Bandpass Filter



Der Tiefpass schneidet alle Frequenzen ab, die oberhalb der Grenzfrequenz liegen und läßt alle Töne durch, die unterhalb liegen. Der Ton klingt voluminös bis dumpf.



Analog dazu läßt der Hochpass nur Frequenzen durch, die oberhalb der Grenzfrequenz liegen. Die darunterliegenden werden abgeschnitten. Der Ton klingt dünn bis scharf.



Der Bandpass läßt nur Frequenzen durch, die oberhalb einer unteren und unterhalb einer oberen Grenzfrequenz liegen. Der Ton klingt hohl oder wie ein Stimmlaut.

Die FILTER–Anweisung legt die Grenzfrequenz, den Filtertyp sowie die Resonanz fest.

Die allgemeine Form lautet:

FILTER [*Freq*][,[*Tp*][,[*Bp*][,[*Hp*][,[*Res*]]]]]

Die Erklärung der einzelnen Parameter:

Freq Ein ganzzahliger Wert zwischen 0 und 2047, der die Filterfrequenz festlegt.

Tp schaltet den Tiefpass
(0 = aus , 1 = an)

Bp schaltet den Bandpass
(0 = aus , 1 = an)

Hp schaltet den Hochpassfilter
(0 = aus , 1 = an)

Res Ein ganzzahliger Wert zwischen 0 und 15, der die Resonanz festlegt.

Ergänzen Sie ihr Programm um folgende Zeile:

```
35 FILTER 1023,1,0,0,10
```

Die Zeile 35 setzt die Grenzfrequenz auf 1023 und schaltet das Tiefpaßfilter ein. Das Hochpaßfilter und der Bandpaß werden ausgeschaltet. Außerdem wird die Resonanz auf 10 eingestellt. Hören sie, wie die Töne klingen.

Jetzt ändern Sie die Zeile 35 wie folgt:

```
35 FILTER 1023,0,1,0,10
```

Dies bewirkt, daß das Tiefpaßfilter aus und gleichzeitig das Bandpaßfilter eingeschaltet wird. Hören Sie sich jetzt noch einmal die Töne an.

Ändern Sie die Zeile 35 wie folgt ab:

```
35 FILTER 1023,0,0,1,10
```

Achten Sie auf den Unterschied, wenn ein Filter nach dem anderen ausgeschaltet wird. Experimentieren Sie noch ein wenig mit den einzelnen Filtern. Hier sind einige Beispiele für Filterbefehle mit anderen Grenzfrequenzen.

```
35 FILTER 2000,0,0,1,15
35 FILTER 1500,0,1,0,5
35 FILTER 1000,1,0,0,8
```

Das erste Beispiel setzt die Grenzfrequenz auf 2000 und wählt die Resonanz 15. Das Hochpaßfilter ist eingeschaltet.
Das zweite Beispiel setzt die Grenzfrequenz auf 1500, legt die Resonanz auf 5 fest und schaltet das Bandpaßfilter ein.
Beim dritten Beispiel ist die Grenzfrequenz auf 1000 festgelegt und die Resonanz auf 8. Das Tiefpaßfilter ist eingeschaltet.

Schreiben eines Musikprogramms

Lassen Sie uns einmal die einzelnen Bestandteile eines Musikprogramms zusammenfügen.

Hier ist ein Beispielprogramm:

```
10 ENVELOPE 9,10,5,10,5,2,4000:PLAY"T9"
20 VOL 8
30 TEMPO 30
35 FILTER 1024,0,1,0,10
40 PLAY"C D E F G A E"
50 PLAY"U5 V1 O5 C D E F G A B"
60 PLAY"U7 V2 O6 X1 C D E F G A B"
70 PLAY"U7 V2 O6 X1 H CD Q# EF I$ GA .S# B"
80 PLAY"U8 V1 O4 X0 .H CD Q# EF I$ GA .S# B"
```

Es folgt eine systematische Erklärung der einzelnen Programmzeilen.

Zeile 10 wählt die Hüllkurve 9. Die PLAY–Anweisung übernimmt die neuen ADSR–Parameter.
Zeile 20 setzt die Lautstärke auf die Stufe 8.
Zeile 30 bestimmt das Tempo.
Zeile 35 filtert die Noten der Zeilen 60 – 70.
Die Grenzfrequenz wird bei 1024 festgelegt. Das Hochpaßfilter wird eingeschaltet und die Resonanzfrequenz bei 10 festgelegt.
Zeile 40 spielt die Noten C D E F G A B (Tonleiter).
Zeile 50 spielt die gleichen Noten wie Zeile 40. Setzt die Oktave und die Lautstärke auf 5 fest.

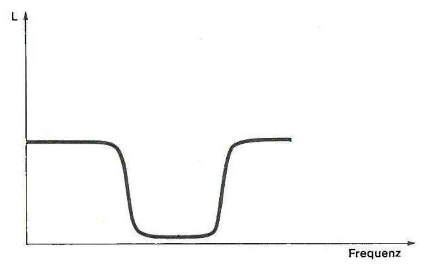
Zeile 60 spielt die gleichen Noten wie in Zeile 40 und 50, nur mit geänderter Lautstärke, anderer Stimme und anderer Oktave. Das Filter ist eigeschaltet. Zeile 70 unterscheidet sich von den vorherigen Zeilen, indem ihre Töne auf halbe, viertel, achtel, sechzehntel Noten geändert werden.
Zeile 80 spielt die gleiche Melodie wie Zeile 70, das Filter ist jedoch ausgeschaltet.

Nun haben Sie alle Informationen, die Sie brauchen, um Ihre eigenen Musikprogramme zu schreiben.

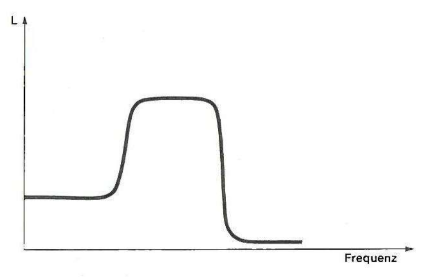
Versuchen Sie einmal, die Noten eines bekannten Musikstücks in den Computer einzugeben und verbinden Sie ein Grafikprogramm mit Ihrem Musikprogramm.

Weitere Möglichkeiten des Filters

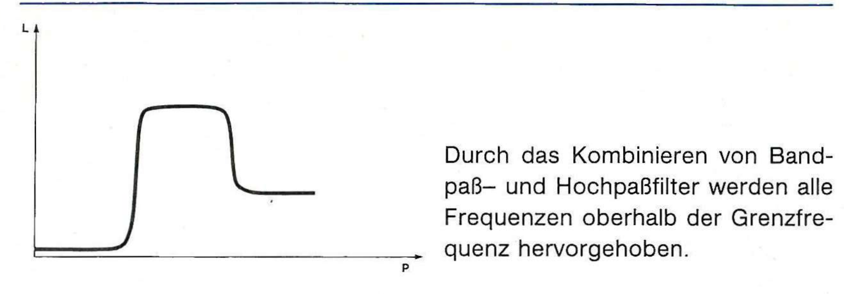
Bisher haben Sie immer nur jeweils eine Filterkombination benutzt. Sie können aber auch alle drei Filtermöglichkeiten kombinieren.



Das Kerbfilter (Notch reject Filter) läßt alle Frequenzen oberhalb und unterhalb der Grenzfrequenzen durch außer den Frequenzen, die im Bereich des Bandpaßfilters liegen.



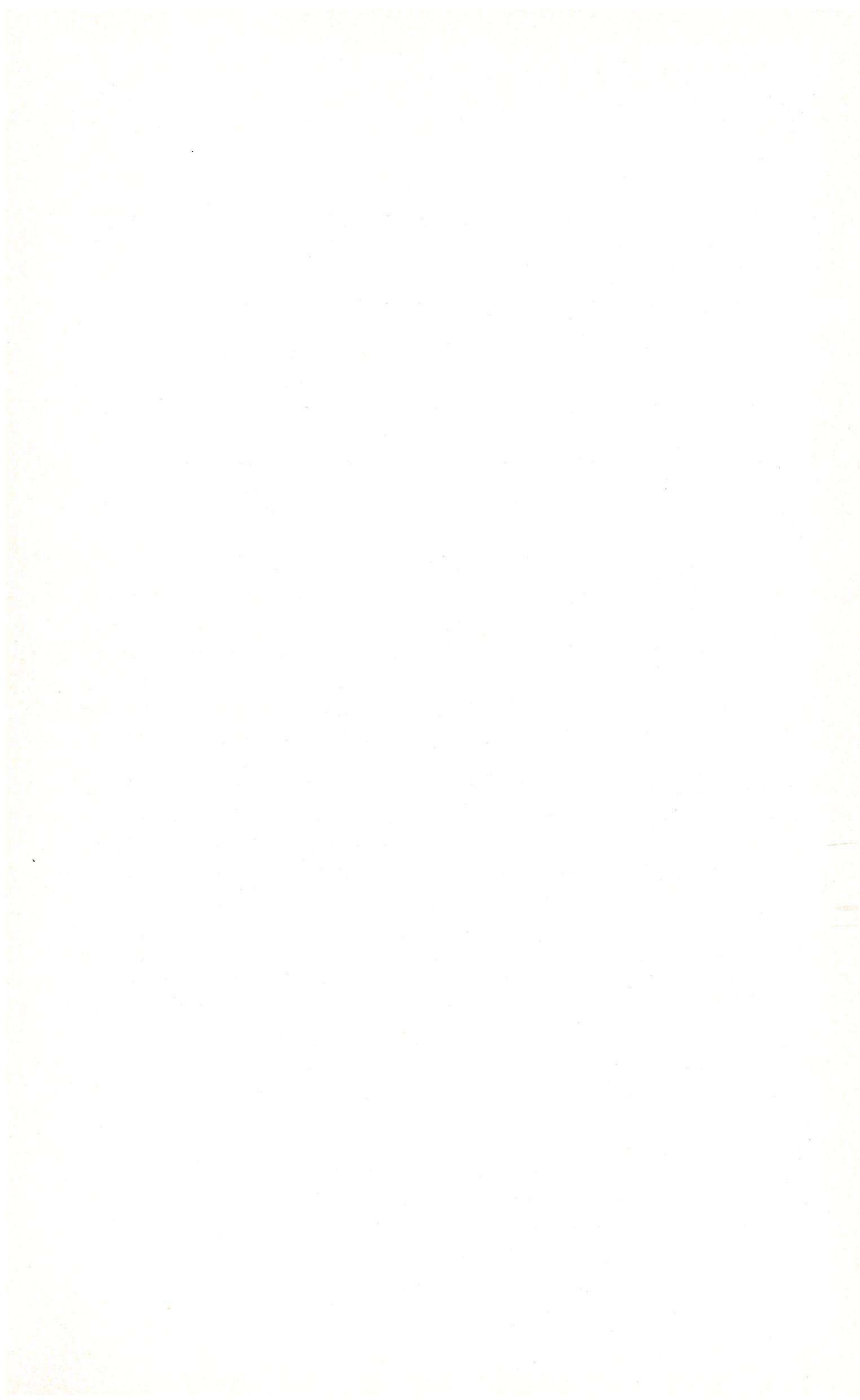
Sie können auch Tiefpaß–, Hochpaß– und Bandpaßfilter miteinander verbinden. Verbinden sie z. B. das Bandpaßfilter mit dem Tiefpaßfilter, so werden alle Frequenzen, die in diesem Bereich liegen, angehoben.



Durch das Kombinieren von Bandpaß– und Hochpaßfilter werden alle Frequenzen oberhalb der Grenzfrequenz hervorgehoben.

Experimentieren Sie einmal mit den Filtern, um die verschiedenen Möglichkeiten der Musikgestaltung kennenzulernen.

5. C64-Modus



5. C64–Modus

Die Tastaturbedienung im C64–Modus ist in Kapitel 3.1 ausführlich beschrieben, so daß hier nicht noch einmal näher darauf eingegangen wird.

Achtung:
Der C64–Modus unterstützt in jedem Fall jedoch nur die ASCII–Tastatur. Wird in den C64–Modus geschaltet, sollte die ASCII/DIN–Taste vorher freigegeben, also entriegelt werden, da sonst die Tastaturbelegung nicht mit der Tastenbeschriftung übereinstimmt.

5.1 BASIC 2.0 im C64–Modus

Dieser Abschnitt ist kein Leitfaden zum Erlernen allgemeiner Programmiertechniken oder der Programmiersprache BASIC, sondern ein Nachschlagewerk für mit allgemeinen Programmierkonzepten vertraute Anwender, die auf den nachfolgenden Seiten alle wissenswerten Fakten für die Erstellung effektiver BASIC–Programme finden. Anfängern wird die Lektüre der in der Commodore–Sachbuchreihe veröffentlichten BASIC–Kurse empfohlen.

In diesem Abschnitt wird der Sprachumfang der Grundversion 2.0 des Commodore–BASIC beschrieben. Diese Version ist sowohl für den C64–Modus als auch für den C128–Modus gültig. Kapitel 4 (C128–Modus) enthält die Beschreibung der Erweiterungen des Commodore–BASIC zur Version 7.0, wie sie nur für den C128–Modus gültig sind.

Da die Sprachstruktur und die Syntax für die nachfolgenden Beschreibungen bereits im Kapitel 4.1 bis 4.4 behandelt wurden, wird hier darauf verzichtet.

5.2 Beschreibung der einzelnen Befehle, Anweisungen, Funktionen und Variablen

Zur Erleichterung des Auffindens eines bestimmten BASIC–Befehls oder einer –Anweisung, –Funktion oder –Variablen sind die nachfolgenden Beschreibungen nicht entsprechend der in Kapitel 4.4 beschriebenen Sprachstruktur von BASIC gegliedert, sondern es sind alle Sprachelemente zusammengefaßt und alphabetisch geordnet.

ABS–Funktion

Format: ***v*= ABS(*x*)**

Zweck: Liefert den Absolutwert des Ausdrucks ***x***.

Beispiel:

```
PRINT ABS(7*(-5))
35
READY.
```

ASC–Funktion

Format: ***v*= ASC(*x$*)**

Zweck: Liefert einen numerischen, ganzzahligen Wert zwischen 0 und 255, der den ASCII–Code des ersten Zeichens der Zeichenkette ***x$*** repräsentiert. Ist ***x$*** eine Zeichenkette der Länge 0 (Leerstring), so wird eine ILLEGAL QUANTITY-Fehlermeldung ausgegeben.

Beispiel:

```
10 A$="TEST"
20 PRINT ASC(A$)
RUN
84
READY.
```

ATN–Funktion

Format: v= ATN(x)

Zweck: Liefert den Arcus Tangens von ***x*** im Bogenmaß im Bereich –PI/2 bis PI/2. Der Ausdruck ***x*** kann von jedem numerischen Typ sein; die Berechnung von ATN erfolgt jedoch auf jeden Fall binär im Gleitkommaformat.

Beispiel:

```
10 INPUT A
20 PRINT ATN(A)
RUN
? 3
1.24904577
READY.
```

CHR$–Funktion

Format: ***v$* = CHR$(*n*)**

Zweck: Liefert eine Ein–Byte–Zeichenkette, deren Zeichen den ASCII–Code ***n*** hat. Deshalb muß ***n*** im Bereich zwischen 0 und 255 liegen. CHR$ wird gewöhnlich dazu verwendet, spezielle Zeichen oder Steuercodes zu erzeugen.

Beispiele:

```
PRINT CHR$(147)
```

Löscht den Bildschirm

```
PRINT CHR$(66)
```

Druckt ein B an der Cursorposition

CLOSE–Anweisung

Format: **CLOSE *Filenummer***

Zweck: Beendet die Ein/Ausgabe über einen Ein–/Ausgabe–Kanal.

Bemerkungen: ***Filenummer*** ist die Nummer zwischen 1 und 255, unter der die Datei mit der OPEN–Anweisung eröffnet wurde.

Der Zusammenhang zwischen einer bestimmten Datei und der Filenummer wird durch die CLOSE–Anweisung aufgehoben. Die Datei kann dann mit der OPEN–Anweisung unter derselben oder einer anderen Filenummer wieder eröffnet werden oder es kann eine beliebige andere Datei unter dieser Filenummer eröffnet werden.

CLOSE auf eine sequentielle Ausgabedatei angewendet, schreibt den letzten Datenpuffer in die Datei und schließt diese mit einer Dateiendemarke ab.

Beispiel:

```
10 OPEN 4,4
20 PRINT#4,"DIESES SIND DRUCKDATEN"
30 CLOSE 4
```

CLR–Befehl

Format: **CLR**

Zweck: Setzt alle numerischen Variablen auf Null, alle Zeichenkettenvariablen auf die Länge 0, leert den Stapelspeicher und den Speicher für Felder und setzt den Zeiger für freien Speicherplatz auf den Wert zurück, der sich aus der Größe des BASIC–Programms ohne alle Variablen ergibt.

Bemerkungen: CLR kann auch innerhalb eines BASIC–Programms ausgeführt werden. Das Programm kann dann fortgesetzt werden, wenn die oben beschriebenen Bedingungen, insbesondere solche, die sich auf GOSUB beziehen, berücksichtigt werden.

Beispiel:

```
X=25
CLR
PRINT X
0
READY.
```

CMD–Befehl

Format: **CMD *Filenummer*[,*Liste von Ausdrücken*]**

Zweck: Adressiert ein Gerät an einer Ein/Ausgabe–Schnittstelle und läßt dieses Gerät nach der Ausgabeoperation im adressierten Zustand.

Bemerkungen: CMD hat dieselbe Parameterliste wie die PRINT#–Anweisung (s. dort).

Beispiel:

```
REM PROGRAMMLISTE AUF DRUCKER AUSGEBEN
OPEN 4,4
CMD4,"PROGRAMMLISTE"
LIST
PRINT#4,"CMD-MODUS WIRD BEENDET"
CLOSE4
```

CONT–Befehl

Format: **CONT**

Zweck: Setzt ein Programm, das durch Drücken der STOP–Taste oder durch die STOP– oder END–Anweisung unterbrochen oder beendet wurde, fort.

Bemerkungen: Die Programmausführung wird unmittelbar an der Stelle, an der die Unterbrechung auftrat, fortgesetzt. Wenn die Unterbrechung nach der Textanzeige einer INPUT–Anweisung durch Drücken der RETURN–Taste ohne vorherige Texteingabe erfolgte, wird das Programm mit der Wiederholung dieser Anzeige (? oder Text) fortgesetzt.

CONT wird üblicherweise in Verbindung mit der STOP-Anweisung zur Fehlersuche in Programmen verwendet. Nach der Programmunterbrechung können Zwischenergebnisse angezeigt oder durch Direkt–Modus–Anweisungen verändert werden. Die Programmausführung wird mit der Eingabe von CONT oder GOTO zusammen mit einer bestimmten Zeilennummer im Direkt–Modus fortgesetzt.

CONT ist ungültig, wenn das Programm mit einer Fehlermeldung abgebrochen wurde oder während der Unterbrechung verändert wurde.

Beispiel: Siehe Beispiel bei der STOP–Anweisung.

COS–Funktion

Format: ***v*= COS(*x*)**

Zweck: Liefert den Cosinus von ***x*** im Bogenmaß. Die Berechnung von COS(*x*) erfolgt binär im Gleitkommaformat.

Beispiel:

```
10 X=2*COS(.4)
20 PRINT X
RUN
1.84212199
READY.
```

DATA–Anweisung

Format: **DATA *Konstantenliste***

Zweck: Speichert numerische und/oder Zeichenketten -Konstanten, auf die mit der READ–Anweisung (s. dort) zugegriffen werden kann.

Bemerkungen: DATA–Anweisungen sind nicht ausführbare Anweisungen, die an beliebiger Stelle im Programm stehen können. Jede DATA–Anweisung kann soviele Konstanten enthalten, wie, getrennt durch Kommata, in eine Befehlszeile (80 Zeichen) passen. Die Zahl der DATA–Anweisungen ist beliebig. Die READ–Anweisung liest die einzelnen DATA–Zeilen in der Reihenfolge ihrer Zeilennummern. Die in diesen Zeilen enthaltenen Daten werden unabhängig von ihrer Zahl und deren Plazierung im Programm als kontinuierliche Elementliste aufgefaßt.

Konstantenliste kann Zeichenketten– und /oder numerische Konstanten jeden Formats, d.h. Zeichenketten sowie Gleitkomma– oder ganze Zahlen enthalten. Numerische Ausdrücke sind nicht erlaubt. Zeichenketten-Konstanten in DATA–Anweisungen müssen nur dann in Anführungsstriche (") eingekleidet werden, wenn sie Kommas, Doppelpunkte oder vor– und/oder nachstehende signifikante Leerstellen enthalten.

Der in der READ–Anweisung deklarierte Variablentyp (numerisch oder Zeichenkette) muß mit dem in den zugehörigen DATA–Anweisungen enthaltenen Konstantentyp übereinstimmen.

Der Lesezeiger kann mit der RESTORE–Anweisung (s. dort) auf den Anfang der ersten DATA–Anweisung gestellt werden.

Beispiel:

```
10 DATA "DAS ","WETTER ","IST ","HEUTE" ,"SCHOEN !"
20 FOR I=1 TO 5
30 READ A$
40 PRINT A$;" ";
50 NEXT
RUN
DAS WETTER IST HEUTE SCHOEN !
READY.
```

DEF FN–Anweisung

Format: **DEF FN *Name*[(*Argumentliste*)] = *Funktionsdefinition***

Zweck: Definiert und benennt eine vom Anwender programmierte BASIC–Funktion.

Bemerkungen: ***Name*** muß ein erlaubter Variablenname sein. Dieser Name, dem FN vorangestellt wird, wird als Name der Funktion betrachtet.

Argumentliste ist das Argument der Funktion, das in der Funktionsdefinition durch eine oder mehrere Gleitkommavariable bezeichnet wird. Letztere werden dann beim Aufruf der Funktion durch die aktuellen Parameter ersetzt.

Funktionsdefinition ist ein beliebiger Ausdruck, der die Operation, die die Funktion ausführen soll, beinhaltet. Die Länge des Ausdrucks ist auf eine BASIC–Anweisungszeile (80 Zeichen) beschränkt. In diesem Ausdruck verwendete Variablennamen dienen nur der formalen Funktionsdefinition und sind nicht mit Programmvariablen desselben Namens zu verwechseln. Ein in einer Funktionsdefinition verwendeter Variablenname kann als Parameter auftreten oder auch nicht. Ist er Parameter, so wird sein Wert beim Aufruf der Funktion ersetzt; andernfalls wird der derzeitige Wert der Variablen verwendet.

Mit DEF FN können keine anwenderspezifischen Zeichenkettenfunktionen definiert werden.

Wenn im Funktionsname ein Variablentyp spezifiziert wird, so wird der Wert des Ausdruckes diesem Typ angepaßt, bevor er der aufrufenden Anweisung übergeben wird. Wenn ein Variablentyp im Funktionsnamen deklariert wurde, der nicht zu dem Typ paßt, den der Ausdruck liefert, so wird eine TYPE MISMATCH–Fehlermeldung ausgegeben.

Die DEF FN–Anweisung muß ausgeführt werden, ehe die dadurch definierte Funktion das erste mal aufgerufen wird, sonst wird eine UNDEFINED FUNCTION–Fehlermeldung ausgegeben.

DEF FN kann nicht im Direkt–Modus verwendet werden.

Beispiel:

```
410 DEF FNAB(X)=X↑3/Y↑3
420 T=FNAB(I)
```

Zeile 410 definiert die Funktion, die in Zeile 420 aufgerufen wird. Dabei wird die Variable X durch den aktuellen Wert von I ersetzt. Die Variable Y behält den ihr zum Zeitpunkt des Funktionsaufrufes zugeordneten Wert.

DIM–Anweisung

Format: **DIM *Liste indizierter Variabler***

Zweck: Definiert die maximale Anzahl von Elementen für Feldvariablen und reserviert den Speicher für die Feldvariable.

Bemerkungen: Wenn ein Feldvariablenname ohne eine vorausgegangene DIM–Anweisung verwendet wird, so ist als maximaler Index 10 erlaubt. Wird ein größerer Index angegeben, so wird eine BAD SUBSCRIPT–Fehlermeldung ausgegeben.

Der kleinste Feldindex ist immer 0. Indizes müssen ganzzahlig sein.

Die DIM–Anweisung setzt alle Elemente des spezifizierten Feldes anfänglich auf Null bzw. Leerstring.

Es können Felder mit bis zu 255 Dimensionen deklariert werden, von denen jedes maximal 32767 Elemente enthalten darf. Auf jeden Fall ist die Feldgröße durch den verfügbaren Speicher begrenzt.

Beispiele:

```
10 DIM A(20)
20 FOR I=0 TO 20
30 READ A(I)
40 NEXT
50 DATA 1,2,3. . .
```

```
10 DIM R3(5,5) : REM 36 ELEMENTE
```

```
10 DIM D$(2,2,2) : REM 27 ELEMENTE
```

END–Anweisung

Format: **END**

Zweck: Beendet den Programmlauf, schließt alle geöffneten Dateien und setzt den Interpreter in den Direkt–Modus.

Bemerkungen: END–Anweisungen zur Programmbeendigung können an jeder Stelle eines Programms stehen.

Die END–Anweisung erzeugt keine Bildschirmmeldung wie z.B. BREAK bei der STOP–Anweisung.

Am Ende eines Programms (letzte Zeile) ist die END–Anweisung wahlfrei.

Beispiel:

```
520 IF K>1000 THEN END
```

EXP–Funktion

Format: *v*= **EXP(*x*)**

Zweck: Liefert die ***x***–te Potenz der Zahl e. ***x*** muß kleiner oder gleich 88.02969191 sein, sonst wird eine OVERFLOW-Fehlermeldung generiert.

Beispiel:

```
10 X=5
20 PRINT EXP(5-1)
RUN
54.5981501
READY.
```

FOR. . .NEXT–Anweisung

Format: **FOR** ***num Var=x*** **TO** ***y*** **[STEP** ***z*****]**

.

.

.

NEXT [***num Var***][,***num Var. . .***]

Zweck: Erlaubt die Mehrfachverarbeitung einer Folge von Befehlen, Anweisungen und/oder Funktionen in einer Programmschleife mit einer definierten Zahl von Durchläufen.

Bemerkungen: ***num Var*** wird als Zähler für die Durchläufe verwendet und muß eine Gleitkommavariable sein. Der erste numerische Ausdruck ***x*** ist der Anfangswert, der zweite numerische Ausdruck ***y*** ist der Endwert des Zählers.

Alle Anweisungen und Programmzeilen nach der FOR-Anweisung bis zur ersten NEXT–Anweisung werden ausgeführt. Dann wird der Zähler um den Wert von ***z*** erhöht und es wird geprüft, ob er größer als der Endwert ***y*** geworden ist. Wenn er nicht grösser ist, verzweigt der Interpreter zurück zu der Anweisung nach der FOR–Anweisung und der Ablauf wird wiederholt. Ist der Zähler grösser als ***y***, so wird das Programm nach der NEXT–Anweisung fortgesetzt. Dies versteht man unter einer FOR. . .NEXT–Schleife.

Wenn für ***z*** ein negativer Wert angegeben ist, so muß der Endwert ***y*** kleiner als der Anfangswert ***x*** sein. ***y*** wird in diesem Fall bei jedem Durchlauf um den Wert von ***z*** vermindert, bis der Zähler kleiner als der Endwert ***y*** wird.

Wird **STEP** ***z*** nicht angegeben, so wird der Zähler bei jedem Durchlauf um 1 erhöht.

FOR. . .NEXT–Schleifen dürfen auch geschachtelt werden, d.h. eine Schleife darf auch innerhalb einer anderen angeordnet sein. Jeder Schleifenzähler muß dann jedoch einen eigenen Namen erhalten. Für alle Zähler in ge-

schachtelten Schleifen reicht eine NEXT–Anweisung, gefolgt von den einzelnen Zählervariablen in der richtigen Reihenfolge und durch Kommas getrennt, wenn die einzelnen NEXT-Anweisungen unmittelbar aufeinander folgen würden.

Die Variablen in der NEXT–Anweisung können weggelassen werden. In diesem Fall bezieht sich jede NEXT–Anweisung auf die zuletzt interpretierte FOR–Anweisung. Findet der Interpreter eine NEXT–Anweisung ohne vorangegangene FOR–Anweisung, so gibt er eine NEXT WITHOUT FOR–Fehlermeldung aus und bricht das Programm ab.

Wegen des begrenzten Stapelspeichers dürfen nur maximal 9 FOR. . .NEXT–Schleifen ineinander geschachtelt werden.

Die Beispiele auf den nächsten Seiten erläutern das Gesagte näher.

Beispiel 1:

```
10 REM GESCHACHTELTE SCHLEIFEN
20 FOR I=1 TO 3:FOR J=1 TO 3:PRINT I;J
30 NEXT J,I
RUN
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
READY.
```

Beispiel 2:

```
10 REM VARIABLENAENDERUNG NACH SETZEN DER
       SCHLEIFE
20 K=10
30 FOR I=1 TO K STEP 2
40 PRINT I;
50 K=K+10
60 PRINT K
70 NEXT
RUN
1 20
3 30
5 40
7 50
9 60
READY.
```

Beispiel 3:

```
10 REM DER ZWEITE WERT IST KLEINER ALS
       DER ERSTE
20 J=0
30 FOR I=1 TO J
40 PRINT I
50 NEXT
RUN
1
READY.
```

In diesem Beispiel wird die Schleife nur einmal durchlaufen, weil der Anfangswert größer als der Endwert ist, was jedoch erst beim Erreichen der NEXT–Anweisung geprüft wird.

Beispiel 4:

```
10 REM GANZZAHL-VARIABLE ALS ZAEHLER
20 FOR I%=1 TO 10:PRINT I%:NEXT
RUN
?SYNTAX ERROR IN 20
READY.
```

Achtung:
Ganzzahl–Variablen als Schleifenzähler sind verboten.

FRE–Funktion

Format: ***v*= FRE(*x*)**

Zweck: Liefert die Anzahl der noch nicht benutzten Bytes im BASIC–Programmspeicher. Für ***x*** kann ein beliebiges Argument angegeben werden, da es keinerlei Wirkung hat. Als Blindargument muß es jedoch vorhanden sein.

Bemerkungen: Da die übergebene Zahl vom Interpreter als ganze Zahl behandelt wird, deren Bereich nur zwischen –32768 und 32767 definiert ist, kann ? FRE(0) auch negative Werte liefern, wenn der verfügbare Speicher mehr als 32767 Bytes beträgt. In diesem Fall berechnet sich der tatsächlich verfügbare Speicher aus der Summe von 65536 und der angezeigten negativen Zahl.

Beispiel:

```
PRINT FRE(0)
1433
READY.
```

GET- und GET#-Anweisungen

Format: **GET**[#***log Dateinr,***]***Variablenliste***

Zweck: Liest ein Zeichen von der Tastatur oder aus einer Datei und weist dieses Zeichen der nächsten Variablen in ***Variablenliste*** zu.

log Dateinr Ein ganzzahliger Wert zwischen 1 und 255, unter der die Datei eröffnet wurde.

Bemerkungen: **GET** ohne die Angabe einer logischen Dateinummer liest aus dem Tastaturpuffer den Code der zuletzt gedrückten Taste und weist ihn der nächsten Variablen (numerisch oder Zeichenkette) in der angegebenen Variablenliste zu. Wurde keine Taste gedrückt, so liefert GET den Wert 0 bzw. eine Zeichenkette der Länge 0 (Leerstring).

GET# liest ein Zeichen aus der unter der logischen Dateinummer eröffneten Datei. Wurde eine Datei mit der Geräteadresse 0 eröffnet, so ist **GET#** identisch mit **GET**, da der Tastatur die Geräteadresse 0 zugeordnet ist.

Beispiel:

```
10 PRINT"WARTEN AUF GEDRUECKTE TASTE"
20 GET A$: IF A$="" THEN 20
```

GOSUB– und RETURN–Anweisungen

Format: **GOSUB *Zeilennummer***

.

.

.

RETURN

Zweck: Verzweigt in ein Unterprogramm, das mit ***Zeilennummer*** beginnt, und kehrt nach Ausführung des Unterprogramms ins Hauptprogramm zurück

Bemerkungen: Die RETURN–Anweisung(en) in einem Unterprogramm bewirken einen Rücksprung zu der Anweisung, die der zuletzt interpretierten GOSUB–Anweisung folgt.

Ein Unterprogramm kann mehrere RETURN–Anweisungen enthalten, wenn der Rücksprung von verschiedenen Bedingungen abhängig sein soll.

Ein Unterprogramm kann an beliebiger Stelle im Programm stehen, muß jedoch vom Interpreter von Hauptprogrammteilen unterschieden werden können. Um unbeabsichtigtes Durchlaufen eines Unterprogramms zu vermeiden, sollte vor dem Unterprogramm eine STOP–, END– oder GOTO-Anweisung stehen, die die Programmsteuerung um das Unterprogramm herumführt. Wenn die Unterprogramme am Anfang des Programms stehen, werden diese schneller ausgeführt.

Unterprogramme können in bis zu 23 Ebenen geschachtelt werden.

Beispiel:

```
10 GOSUB 40
20 PRINT"AUS DEM UNTERPROGRAMM ZURUECK"
30 END
40 PRINT"IM UNTERPROGRAMM"
50 RETURN
RUN
IM UNTERPROGRAMM
AUS DEM UNTERPROGRAMM ZURUECK
READY.
```

GOTO–Anweisung

Format: **GOTO *Zeilennummer***

Zweck: Verzweigt unbedingt aus der normalen Programmabfolge zu einer spezifizierten Zeilennummer.

Bemerkungen: Wenn ***Zeilennummer*** eine Zeile mit einer ausführbaren Anweisung kennzeichnet, werden diese und die darauf folgenden Zeilen abgearbeitet.

Existiert die spezifizierte Zeile nicht, so wird die Fehlermeldung UNDEF'D STATEMENT ERROR angezeigt.

Beispiel:

```
10 READ R
20 PRINT"R = ";R,
30 A=3.14*R↑2
40 PRINT"FLAECHE =";A
50 GOTO 10
60 DATA 5,7,12
RUN
R = 5         FLAECHE = 78.5
R = 7         FLAECHE = 153.86
R = 12        FLAECHE = 452.16
?OUT OF DATA ERROR IN 10
READY.
```

IF–Anweisung

Format: **IF *Ausdruck* THEN *Anweisungen***
IF *Ausdruck* GOTO *Zeilennummer*

Zweck: Erlaubt die Verzweigung zu anderen Anweisungen oder in verschiedene Programmteile, abhängig vom logischen Wahrheitgehalt eines numerischen Ausdrucks.

Bemerkungen: Wenn das Ergebnis von ***Ausdruck*** logisch "wahr", also von Null verschieden ist, wird die THEN– oder GOTO-Klausel ausgeführt.

THEN kann entweder von einer Zeilennummer zum Verzweigen oder von einer oder mehrere Anweisungen gefolgt werden.

GOTO wird immer von einer Zeilennummer gefolgt. Ist das logische Ergebnis von ***Ausdruck*** "falsch", also Null, so wird die THEN– oder GOTO–Klausel ignoriert und das Programm wird mit der folgenden Befehlszeile fortgesetzt.

IF. . .THEN–Anweisungen können auch geschachtelt werden, wobei die Schachtelungen nur durch die Befehlszeilenlänge (80 Zeichen) begrenzt werden:

IF A=B THEN IF B=C THEN PRINT"A=C"

Wird eine im Direkt–Modus eingegebene IF. . .THEN–Anweisung von einer Zeilennummer gefolgt, so generiert der Interpreter eine UNDEFINED STATEMENT–Fehlermeldung, selbst wenn vorher eine Zeile mit dieser Nummer eingegeben wurde.

Bei Verwendung von IF zum Testen eines Wertes, der sich aus einer Gleitkommaberechnung ergeben hat, ist zu beachten, daß die interne Darstellung des Wertes ungenau sein kann. Deshalb sollte ein Test immer für den Bereich gemacht werden, innerhalb dessen die Genauigkeit variiert. Es sollte also z.B. beim Testen einer Varia-

blen auf den berechneten Wert 1.0 folgendermassen verfahren werden:

IF ABS(A–1.0) < = 1.0E–6 THEN . . .

Dieser Test liefert das Ergebnis "wahr", wenn der Wert von A gleich 1.0 mit einem relativen Fehler von weniger als 1.06E–6 ist.

Beispiel 1:

```
100 IF I THEN GET I
```

Diese Anweisung prüft auf eine gedrückte Taste, falls der Wert von I nicht Null ist.

Beispiel 2:

```
100 IF(I>10)AND(I<20)THEN
      DB=1979-1:GOTO 300
110 PRINT"BEREICHSUEBERSCHREITUNG"
```

Wenn I größer als 10 und kleiner als 20 ist, wird DB berechnet und das Programm wird mit Zeile 300 fortgesetzt. Sonst wird Zeile 110 ausgeführt.

INPUT–Anweisung

Format: **INPUT** ["***Text***";]***Variablenliste***

Zweck: Erlaubt Dateneingabe über die Tastatur während der Programmausführung. Die Dateneingabe wird durch Drücken der RETURN–Taste beendet.

Bemerkungen: Trifft der Interpreter auf eine INPUT–Anweisung, so wird der Programmlauf angehalten, ein Fragezeichen wird auf dem Bildschirm ausgegeben, um zu zeigen, daß das Programm Dateineingabe erwartet und der Cursor blinkt.

Wurde ***Text*** spezifiziert, so wird dieser vor dem Fragezeichen abgebildet. Die geforderten Daten können dann über die Tastatur eingegeben werden. Diese Daten werden der (den) Variablen in der ***Variablenliste*** zugeordnet; deshalb muß die Zahl der DatenEinheiten (getrennt durch Kommas) mit der Zahl der Variablen in der Liste übereinstimmen.

Die Variablen in der Liste dürfen Namen für numerische und Zeichenketten–Variablen (auch Feldvariablen) sein. Der Typ jeder eingegebenen Dateneinheit muß mit dem Typ der korrespondierenden Variablen übereinstimmen.

Eingegebene Zeichenketten müssen nicht in Anführungsstriche (") eingekleidet werden, es sei denn, sie enthalten Kommas und/oder Doppelpunkte.

Die Dateneingabe ist auf die Länge einer logischen Bildschirmzeile (80 Zeichen) begrenzt. Wegen des Fragezeichens können also höchstens 78 Zeichen eingegeben werden. Wird diese Zahl überschritten, nimmt INPUT die zuletzt eingegebene logische Bildschirmzeile als gesamte Eingabe. Als logische Bildschirmzeile werden Bildschirmdaten von bis zu 80 Zeichen, gerechnet vom Zeilenanfang bis zum Wagenrücklauf–Code (RETURN-Taste), betrachtet.

Wird bei INPUT der falsche Datentyp eingegeben, also z.B. Zeichenkettendaten anstatt numerischer, so wird die Meldung ?REDO FROM START ausgegeben und es wird auf die richtige Eingabe gewartet.

Im Direkt–Modus ist INPUT nicht erlaubt. In diesem Fall wird die ILLEGAL DIRECT–Fehlermeldung ausgegeben.

Werden bei INPUT zuviele Dateneinheiten eingegeben, wird die EXTRA IGNORED–Meldung ausgegeben. Bei zuwenigen Dateneinheiten werden die fehlenden mit der Anzeige ?? nachgefordert.

Beispiel 1:

```
10 REM EINGABE VON ZUVIELEN ZEICHEN
20 INPUT A$
30 PRINT:PRINT A$
RUN
? DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
****

****
```

Es wurde 78 mal D und 4 mal * eingegeben. Da INPUT bei Überschreitung der Maximallänge nur die letzte logische Bildschirmzeile berücksichtigt, wird nur **** der Stringvariablen A$ zugeordnet.

Beispiel 2:

```
10 INPUT X
20 PRINT X"ZUM QUADRAT IST"X↑2
RUN
?5
5 ZUM QUADRAT IST 25
READY.
```

Beispiel 3:

```
10 PI=3.14159265
20 INPUT"RADIUS EINGEBEN";R
30 A=PI*R↑2
40 PRINT"KREISFLAECHE IST";A
50 PRINT
60 GOTO 20
RUN
RADIUS EINGEBEN? 7.4
KREISFLAECHE IST 172.033614

RADIUS EINGEBEN? usw.
```

INPUT#–Anweisung

Format: **INPUT#*log Dateinr,Variablenliste***

Zweck: Liest Daten aus einer sequentiellen oder Relativ–Datei und weist sie Programmvariablen zu.

Bemerkungen: ***log Dateinr*** ist die Nummer, unter der die Datei zur Eingabe mit der OPEN–Anweisung (s. dort) eröffnet wurde.

Variablenliste enthält die Namen der Variablen, denen die Datenelemente aus der Datei zugewiesen werden. Die Datentypen müssen den Variablentypen entsprechen.

INPUT# gibt kein ? als Anzeige aus, wenn als Eingabegerät die Tastatur gewählt wurde.

Die Dateneinheiten in der Datei müssen genauso angeordnet sein, als würden sie über die Tastatur eingegeben. Vor– und nachlaufende Leerstellen, Wagenrücklauf– und Zeilenvorschub–Codes werden ignoriert. Trennzeichen zwischen Variableninhalten können Komma oder Doppelpunkt sein; der Wagenrücklaufcode trennt auf jeden Fall einzelne Datenelemente voneinander.

Bei Nichtübereinstimmung von Daten– und Variablentyp wird eine FILE DATA ERROR–Fehlermeldung ausgegeben. Beim Versuch, Daten am Dateiende zu lesen oder bei Zeitüberschreitung an der Eingabeschnittstelle (s. STATUS–Systemvariable) liefert INPUT# einen Wagenrücklauf–Code (CHR$(13)).

INPUT# kann maximal 80 Zeichen lesen.

Beispiel:

```
10 REM LESEN VON KASSETTE BIS FILEENDE
20 OPEN 3,1,0
30 INPUT#3,A$
40 IF STATUS AND 64 THEN FE=1
50 PRINT A$
60 IF FE THEN CLOSE 3:END
70 GOTO 30
```

Die Abfrage des Rechner–Statuswortes (hier in Zeile 40) wird bei STATUS in diesem Kapitel ausführlich beschrieben.

INT–Funktion

Format: *v*= **INT(*x*)**

Zweck: Liefert die größte ganze Zahl, die kleiner oder gleich ***x*** ist.

Beispiel:

```
PRINT INT(99.89),INT(-12.11)
99        -13
READY.
```

LEFT$–Funktion

Format: ***v$*** **= LEFT$(*x$,n*)**

Zweck: Übergibt eine Zeichenkette, die aus den ***n*** linken Zeichen von ***x$*** besteht. ***n*** muß im Bereich zwischen 0 und 255 liegen. Wenn ***n*** größer als die Länge von ***x$*** ist, wird die gesamte Zeichenkette ***x$*** geliefert. Wenn ***n*** Null ist, dann wird eine Zeichenkette der Länge 0 (Leerstring) übergeben.

Beispiel:

```
10 A$="COMMODORE BUEROMASCHINEN"
20 B$=LEFT$(A$,9)
30 PRINT B$
RUN
COMMODORE
READY.
```

LEN–Funktion

Format: ***v*= LEN(*x$*)**

Zweck: Übergibt die Anzahl der Zeichen in der Zeichenkette ***x$***. Es werden alle Zeichen, also auch die nicht abdruckbaren und Leerzeichen, gezählt.

Beispiel:

```
10 X$="COMMODORE BRAUNSCHWEIG"+CHR$(13)
20 PRINT LEN(X$)
RUN
23
READY.
```

LET-Anweisung

Format: [**LET**] ***Variable = Ausdruck***

Zweck: Weist den Wert eines Ausdrucks einer Variablen zu.

Bemerkungen: Das Wort LET ist wahlfrei, d.h. bei der Zuweisung eines Wertes zu einer Variablen genügt das Gleichheitszeichen.

Beispiel:

```
110 LET D=12:LET E=12*2
120 LET F=144/12
130 LET SUM=D+E+F
```

ist gleichbedeutend mit:

```
110 D=12:E=12*2
120 F=144/12
130 SUM=D+E+F
```

LIST–Befehl

Format: **LIST** [***Zeile1***][–[***Zeile2***]]

Zweck: Listet einen Teil oder ein ganzes Programm auf das gegenwärtig aktivierte Ausgabegerät (Bildschirm, Drukker, Kassette, Floppy Disk).

Bemerkungen: Der BASIC–Interpreter wird nach der Ausführung von LIST immer in den Direkt–Modus gesetzt.

Mit LIST ohne Zeilennummernangabe wird das ganze Programm beginnend mit der kleinsten Zeilennummer gelistet. Das Listen wird entweder durch das Programmende oder durch Drücken der STOP–Taste beendet. Wird nur ***Zeile1*** angegeben, so wird nur diese eine Zeile gelistet.

Beispiele:

`LIST`	Listet das gesamte im Speicher befindliche Programm.
`LIST 500`	Listet Zeile 500.
`LIST 150-`	Listet alle Zeilen von Zeile 150 einschließlich bis Programmende.
`LIST -1000`	Listet alle Zeilen vom Anfang des Programms bis Zeile 1000 einschließlich.
`LIST 150-700`	Listet alle Zeilen von Zeile 150 bis Zeile 700 einschließlich.

LOAD–Befehl

Format: **LOAD "*Dateiname*"**[,***Geräteadresse***]
[***Verschieb***]

Zweck: Lädt eine Programmdatei von einem externen Speichergerät (Kassette, Floppy Disk) in den Speicher des Rechners.

Bemerkungen: ***Dateiname*** ist der Name, unter dem das Programm mit der SAVE–Anweisung (s. dort) auf ein externes Speichergerät gespeichert wurde.

Geräteadresse muß ein ganzzahliger Wert zwischen 1 und 15 sein. Wird ***Geräteadresse*** weggelassen, so wird das Programm von Gerät 1 (Kassettenstation) geladen.

Verschieb Ein ganzzahliger Wert, der angibt, ob das Programm an den Anfang des BASIC–Programmspeichers (0) oder an die Adresse, mit der es auch auf Kassette oder Floppy–Disk gespeichert wurde (1), geladen wird. Voreingestellt ist hier 0. Der Wert dieses Parameters muß immer 1 sein, wenn Maschinensprache-programme geladen werden sollen.
Durch LOAD werden alle eröffneten Dateien geschlossen sowie alle gesetzten Variablen und das ggfs. im Speicher befindliche Programm gelöscht, ehe das spezifizierte Programm geladen wird.

Wenn LOAD innerhalb eines Programms ausgeführt wird, wird das dadurch geladene BASIC–Programm sofort gestartet, wobei alle eröffneten Dateien offen bleiben. Dadurch können mehrere Programme oder Programmsegmente miteinander verkettet werden. Gesetzte Variablen bleiben durch die Verkettung erhalten.

Das nachgeladene Programm darf jedoch höchstens genau so groß sein, wie das aufrufende.

Beispiele: Laden eines Programms von Kassette:

```
LOAD "TESTPRG"
PRESS PLAY ON TAPE
```

Laden eines Maschinenspracheprogramms von Diskette:

```
LOAD "MULT",8,1
```

LOG–Funktion

Format: ***v*= LOG(*x*)**

Zweck: Liefert den natürlichen Logarithmus von ***x.x*** muß größer als Null sein; andernfalls wird eine ILLEGAL QUANTITY-Fehlermeldung ausgegeben.

Beispiel:

```
PRINT LOG(45/7)
1.86075234
READY.
```

MID$–Funktion

Format: ***v$*** = MID$(***x$***,***n***[,***m***])

Zweck: Liefert eine Teilzeichenkette von ***x$*** mit ***m*** Zeichen, beginnend beim ***n***–ten Zeichen von ***x$***. ***n*** und ***m*** müssen im Bereich zwischen 0 und 255 liegen. Wird ***m*** weggelassen, oder sind rechts vom ***n***–ten Zeichen weniger als ***m*** Zeichen in ***x$*** vorhanden, so werden ab dem ***n***–ten Zeichen alle rechten Zeichen von ***x$*** geliefert. Ist ***n*** größer als die Länge von ***x$***, so wird eine Zeichenkette der Länge 0 (Leerstring) übergeben.

Beispiel:

```
10 A$="GUTEN"
20 B$="MORGEN ABEND MITTAG"
30 PRINT A$;MID$(B$,8,5)
RUN
GUTEN ABEND
READY.
```

NEW–Befehl

Format: **NEW**

Zweck: Löscht das gegenwärtig im Speicher befindliche Programm sowie alle Variablen.

Bemerkungen: NEW wird gewöhnlich im Direkt–Modus eingegeben, ehe ein neues Programm ediert wird. Der Interpreter setzt den Rechner auf jeden Fall nach Ausführen von NEW in den Direkt–Modus.

ON. . .GOSUB– und ON. . .GOTO–Anweisungen

Format: **ON *Ausdruck* GOTO *Zeilennummernliste***
ON *Ausdruck* GOSUB *Zeilennummernliste*

Zweck: Verzweigt zu einer von mehreren spezifizierten Zeilennummern in Abhängigkeit des Wertes, der von ***Ausdruck*** geliefert wird.

Bemerkungen: Der Wert von ***Ausdruck*** bestimmt, zu welcher Zeilennummer aus der Liste das Programm verzweigt. Wenn der Wert z.B. 3 ist, so stellt die 3. Zeilennummer in der Liste das Sprungziel dar.

Vor der Verzweigung wird der Wert auf jeden Fall in eine ganze Zahl umgewandelt, d.h., ggfs. vorhandene Dezimalstellen werden abgeschnitten.

Bei der ON. . .GOSUB–Anweisung muß jede spezifizierte Zeilennummer die Anfangszeile eines Unterprogramms kennzeichnen.

Ist der Wert des Ausdrucks negativ, so wird eine ILLEGAL QUANTITY–Fehlermeldung ausgegeben. Ist er Null oder größer als die Anzahl der in der Liste angegebenen Zeilennummern, so wird das Programm mit der auf diese Anweisung folgenden Zeile fortgesetzt.

Beispiel:

```
100 ON L-1 GOTO 150,300,320,220
```

Für L=2 wird nach Zeile 150 verzweigt, für L=3 nach Zeile 300 usw.

OPEN–Anweisung

Format: **OPEN *log Dateinr*[,*Geräteadresse***
[,*Sekundäradr*[,"*Dateiname*"]]]

Zweck: Eröffnet eine Ein/Ausgabe–Datei bzw. einen Ein–/Ausgabekanal für ein Gerät.

Bemerkungen: ***log Dateinr*** muß zwischen 1 und 255 liegen.

Falls keine ***Geräteadresse*** angegeben wird, wird 1 angenommen (Kassettenstation).

Für ***Dateiname*** kann eine bis zu 16 Zeichen lange Zeichenkette angegeben werden.

Im Falle der Ein–/Ausgabe über die IEEE 488–Schnittstelle (seriell oder parallel) werden mit jeder GET#–, INPUT#– oder PRINT#–Anweisung die Geräteadresse und eine ggfs. spezifizierte Sekundäradresse über die Schnittstelle gesendet.

Bei Floppy–Disk–Dateien wird der Dateityp mit P (Programmdatei) angenommen, falls nicht S oder U (sequentielle Datendatei) getrennt durch Komma an den Dateinamen angefügt wurde. Wird nach einem weiteren Komma ein W angegeben, so wird die spezifizierte Datei zum Schreiben eröffnet, andernfalls zum Lesen.

Bei Kassettendateien bezeichnet eine Sekundäradresse 0 eine Eingabedatei, 1 eine Ausgabedatei und 2 eine Ausgabedatei, bei der mit der CLOSE–Anweisung (s. dort) eine Bandendemarke hinter die Datei geschrieben wird.

Dateien bzw. Datenkanäle können für Tastatur (Geräteadr. 0), Kassettengerät (Geräteadr. 1) oder Bildschirm (Geräteadr. 3) eröffnet werden.

Geräteadressen größer als 3 beziehen sich auf Geräte, die an die IEEE488–Schnittstelle (seriell oder parallel) angeschlossen werden können (z.B. 4 für Drucker, 8 für Floppy Disk).

Beispiel:

```
10 REM EROEFFNEN EINES KASSETTEN-
15 REM SCHREIBFILES
20 REM ANSCHLIESSENDER BANDENDEMARKE
30 OPEN 6,1,2,"DATENFILE"
40 FOR I=1 TO 10
50 PRINT#6,CHR$(I)
60 NEXT
70 CLOSE 6
```

Weitere Beispiele für die Dateieröffnung bei Floppy-Disk-Betrieb sind im Kapitel 6.6 beschrieben.

PEEK–Funktion

Format: ***v*= PEEK(*n*)**

Zweck: Liefert den Inhalt der Speicherzelle mit der Adresse ***n*** als ganzzahligen Wert zwischen 0 und 255. Für ***n*** müssen ganzzahlige Werte zwischen 0 und 65535 angegeben nehmen. PEEK ist das Gegenstück zur POKE–Anweisung (s. dort).

Beispiel:

```
A=PEEK(53281)AND15
```

Der Wert der Variablen A ist nach dieser Anweisung der Code für die gegenwärtig eingestellte Hintergrundfarbe (C64–Modus) des Bildschirms.

POKE–Anweisung

Format: **POKE *n,m***

Zweck: Schreibt eine 8–Bit–Binärinformation in eine spezifizierte Speicherzelle.

Bemerkungen: Der Wert des ganzzahligen Ausdrucks ***n*** bezeichnet die zu beschreibende Speicherzelle; der Wert des ganzzahligen Ausdrucks ***m*** bezeichnet das zu speichernde Datum.

n muß im Bereich zwischen 0 und 65535 und ***m*** im Bereich zwischen 0 und 255 liegen.

Das Gegenstück zur POKE–Anweisung ist die PEEK-Funktion (s. dort), deren Argument eine Speicherzelle bezeichnet, deren Inhalt ausgelesen werden soll.

POKE und PEEK sind effektive Hilfsmittel für die Datenspeicherung, das Laden von Unterprogrammen in Maschinensprache sowie die Übergabe von Parametern und Ergebnissen zwischen BASIC–Hauptprogrammen und Maschinensprache–Unterprogrammen.

Beispiel 1:

```
10 REM FARB- UND ZEICHENSTEURUNG
20 POKE 1024,1:POKE 55296,6
```

Hier wird der Buchstabe A in blauer Farbe in der HOME-Position des Bildschirms abgebildet (C64–Modus).

Beispiel 2:

```
10 REM ALLE TASTEN HABEN WIEDERHOLFUNKT.
20 POKE 650,128
30 REM NUR DIE CURSORSTEUERTASTEN HABEN
40 REM WIEDERHOLFUNKTION
50 POKE 650,0
```

POS–Funktion

Format: ***v*=POS(*n*)**

Zweck: Liefert die gegenwärtige Spaltenposition des Cursors auf dem Bildschirm. Die äußerst linke Spalte ist die Position 0.

n ist ein Blindargument, das aus formalen Gründen jedoch angegeben werden muß.

Beispiel:

```
IF POS(0)>20 THEN PRINTCHR$(13)
```

PRINT– und PRINT#–Anweisungen

Format: **PRINT**[#***log Dateinr***,][***Liste von Ausdr***]

Zweck: Gibt Daten an den Bildschirm oder über einen spezifizierten Ausgabekanal aus.

Bemerkungen: Wird ***Liste von Ausdr*** nicht angegeben, so wird eine Leerzeile gedruckt. Andernfalls werden die Ausdrücke ausgewertet und deren Werte an das in der zugehörigen OPEN–Anweisung unter ***log Dateinr*** spezifizierte Ausgabegerät oder die Ausgabedatei ausgegeben.

Es sind numerische und/oder Zeichenkettenausdrücke erlaubt.

Zeichenkettenkonstanten müssen in Anführungsstriche (") eingekleidet werden.

Die Position jedes zu druckenden Datums (Datum ist die Einzahl von Daten) wird durch die Interpunktion, die die Daten in der Liste voneinander trennt, bestimmt. Der BASIC–Interpreter teilt die Druckzeile in Druckzonen von je 10 Leerstellen ein. Ein Komma in der Liste von Ausdrücken bewirkt, daß der Wert des darauffolgenden Ausdrucks ab dem Anfang der nächsten Druckzone gedruckt wird, wohingegen ein Semikolon bewirkt, daß der nächste Wert unmittelbar hinter den vorausgehenden Wert gedruckt wird.

Ein Komma oder Semikolon am Ende einer Liste von Ausdrücken bedeutet, daß die Werte der nächsten PRINT–Anweisung in der nächsten Druckzone derselben Zeile oder unmittelbar anschließend in derselben Zeile gedruckt werden. In beiden Fällen wird ein Wagenrücklauf–Code unterdrückt.

Ist die zu druckende Zeile länger als 40 Zeichen (oder wahlweise 80 beim C128–Modus), so wird die Ausgabe in der nächsten physikalischen Zeile fortgesetzt.

Gedruckten Zahlenwerten folgt immer eine Leerstelle. Positiven Zahlenwerten ist eine Leerstelle, negativen ein Minus–Zeichen vorangestellt. Jede Zahl zwischen 0 und 0.01 wird in der wissenschaftlichen Exponentialdarstellung (s. Kapitel 2.4) wiedergegeben. Die PRINT–Anweisung (**nicht** PRINT #) kann durch ? abgekürzt werden.

Beispiel 1:

```
10 X=5
20 PRINT X+5,X-5,X*(-5),X↑5
RUN
10        0         -25       3125
READY.
```

Die Kommata zwischen den Ausdrücken in Zeile 20 bewirken, daß jeder Wert an den Anfang einer 10 Leerstellen breiten Druckzone gedruckt wird.

Beispiel 2:

```
10 INPUT X
20 PRINT X"HOCH 2 IST"X↑2"UND";
30 PRINT X"HOCH 3 IST"X↑3
40 PRINT
50 GOTO 10
RUN
? 9
9 HOCH 2 IST 81 UND 9 HOCH 3 IST 729

? usw.
```

Hier bewirkt das Semikolon am Ende von Zeile 20, daß die Werte beider PRINT–Anweisungen in den Zeilen 20 und 30 in dieselbe Zeile gedruckt werden. Zeile 40 bewirkt das Drucken einer Leerzeile.

Beispiel 3:

```
10 FOR X=1 TO 5
20 J=J+5
30 K=K+10
40 ?J;K;
50 NEXT
RUN
5 10 10 20 15 30 20 40 25 50
READY.
```

Bei diesem Beispiel wurde in Zeile 40 für die PRINT–Anweisung das ?–Zeichen gewählt. Die Semikolons bewirken das Drucken der einzelnen Werte unmittelbar hintereinander getrennt durch 2 Leerstellen (jede Zahl wird von einer Leerstelle gefolgt, positiven Zahlen ist eine Leerstelle vorangestellt). Wird das Programm mit LIST ausgelistet, so wird das ?–Zeichen in Zeile 40 durch das Wort PRINT ersetzt.

READ–Anweisung

Format: **READ *Variablenliste***

Zweck: Liest Daten aus einer DATA–Anweisung (s. dort) und weist sie Variablen zu.

Bemerkungen: Eine READ–Anweisung kann nur in Verbindung mit einer DATA–Anweisung benutzt werden.

Jeder Variablen aus der Liste, die eine numerische oder eine Zeichenketten–Variable sein kann, wird immer nur ein Wert aus der DATA–Anweisung zugewiesen. Daten– und Variablentypen müssen übereinstimmen. Andernfalls wird eine SYNTAX ERROR–Fehlermeldung ausgegeben.

Eine einzelne READ–Anweisung kann sequentiell auf mehrere DATA–Anweisungen zugreifen wie auch mehrere READ–Anweisungen auf eine DATA–Anweisung zugreifen können.

Wenn die Anzahl der Variablen in der Variablenliste größer ist als die Anzahl von Elementen in der (den) DATA–Anweisung(en), wird eine OUT OF DATA–Fehlermeldung ausgegeben. Sind weniger Variablen in der Liste spezifiziert, als Elemente in der (den) DATA–Anweisung(en) vorhanden sind, so lesen folgende READ-Anweisungen die noch nicht gelesenen Elemente. Folgen in einem solchen Fall keine weiteren READ–Anweisungen, so bleiben überzählige Datenelemente unberücksichtigt. Um DATA–Anweisungen wiederholt von Anfang an zu lesen, kann die RESTORE–Anweisung (s. dort) verwendet werden.

Beispiel 1:

```
80 FOR I=1 TO 10
90 READ A(I)
100 NEXT
110 DATA 3.08,5.19,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.37
.
.
.
```

In diesem Programmsegment werden die Elemente der DATA–Anweisungen der Zeilen 110 und 120 in das Feld A gelesen.

Beispiel 2:

```
10 PRINT "PLZ","STADT","LAND"
20 READ PZ,S$,L$
30 DATA 6000,"FRANKFURT","HESSEN"
40 PRINT PZ,S$,L$
RUN
PLZ         STADT       LAND
6000        FRANKFURT   HESSEN
READY.
```

Hier werden numerische und String–Daten aus der DATA–Anweisung in Zeile 30 gelesen und ausgedruckt.

REM–Anweisung

Format: **REM** [*Kommentar*]

Zweck: Mit dieser Anweisung können erläuternde Kommentare in ein Programm eingefügt werden.

Bemerkungen: REM–Anweisungen werden nicht ausgeführt, jedoch exakt wiedergegeben, wenn das Programm gelistet wird.

Von einer GOTO– oder GOSUB–Anweisung kann zu einer REM–Anweisung verzweigt werden. Das Programm wird dann mit der nächsten auf die REM–Anweisung folgenden ausführbaren Anweisung fortgesetzt.

Der Text nach REM sollte keine Zeichen enthalten, die bei gedrückter SHIFT–Taste eingegeben wurden, da diese beim LISTen als BASIC–Schlüsselwörter interpretiert und ausgedruckt werden.

Beispiel:

```
10 REM BERECHNUNG DER MITTLEREN
15 REM GESCHWINDIGKEIT
20 FOR I=1 TO 20
30 SUM=SUM+V(I)
40 NEXT
50 VM=SUM/(I-1)
```

RESTORE–Anweisung

Format: **RESTORE**

Zweck: Setzt den Lesezeiger der READ–Anweisung auf den Anfang der **ersten** DATA–Anweisung im Programm.

Bemerkungen: Nach der Ausführung einer RESTORE–Anweisung greift die nächste READ–Anweisung auf das erste Datenelement in der ersten DATA–Anweisung im Programm zu.

Beispiel:

```
10 READ A,B,C
20 PRINT A,B,C
30 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 DATA 57,68,79
RUN
57          68          79
57          68          79
READY.
```

RIGHT$–Funktion

Format: ***v$*** = RIGHT$(***x$***,***n***)

Zweck: Liefert die rechten ***n*** Zeichen aus der Zeichenkette ***x$***. Wenn ***n*** gleich oder größer der Länge von ***x$*** ist, wird ***x$*** übergeben. Für ***n*** = 0 wird eine Zeichenkette der Länge 0 (Leerstring) übergeben. ***n*** muß einen Wert zwischen 0 und 255 haben.

Beispiel:

```
10 A$="COMMODORE BUEROMASCHINEN"
20 PRINT RIGHT$(A$,14)
RUN
BUEROMASCHINEN
READY.
```

RND–Funktion

Format: ***v*=RND(*x*)**

Zweck: Liefert eine Zufallszahl zwischen 0 und 1, die, abhängig vom Argument ***x***, unterschiedlich erzeugt wird:

x*>0:** Es wird immer der nächste Wert einer Zufallszahlenreihe, die durch einen numerischen Algorithmus im BASIC–Interpreter berechnet wird, geliefert. Die Reihe ist vom Wert des Argumentes ***x unabhängig und wird beim Einschalten des Rechners durch einen zufälligen Anfangswert initialisiert.

x*<0:** Jedes Argument ***x initialisiert eine neue Zufallszahlenreihe. Gleiche Argumente führen zu gleichen Zufallszahlenreihen.

***x*=0:** Aus verschiedenen, von einander unabhängigen Zeitgebern wird durch einen Algorithmus eine Zufallszahl erzeugt.

Beispiel:

```
10 FOR I=1 TO 5
20 PRINT INT(RND(X)*100);
30 NEXT
RUN
24 30 83 45 1
READY.
```

RUN–Befehl

Format: **RUN** [*Zeilennummer*]

Zweck: Startet das gegenwärtig im Programmspeicher befindliche BASIC–Programm.

Bemerkungen: Wird ***Zeilennummer*** angegeben, so wird das Programm mit der dadurch bezeichneten Zeile gestartet. Andernfalls beginnt die Ausführung mit der niedrigsten Zeilennummer.

Vor dem Programmstart wird durch RUN zuerst CLR (s. dort) ausgeführt.

Ein Programm wird durch Rücksetzen des Interpreters in den Direkt–Modus beendet, wenn:

1. keine ausführbaren Zeilen mehr vorhanden sind.
2. eine END– oder STOP–Anweisung ausgeführt wurde.
3. ein Fehler während der Ausführung auftritt.

SAVE–Befehl

Format: **SAVE** ["***Dateiname***"[,***Geräteadr*** [,***Option***]]]

Zweck: Speichert eine BASIC–Programmdatei auf einem spezifizierten Ausgabegerät.

Bemerkungen: Wenn ***Geräteadr*** nicht angegeben wird, so wird das im Programmspeicher befindliche BASIC–Programm auf Kassette (Geräteadr. 1) gespeichert.

Als ***Option*** kann bei Speicherung auf Kassette eine Null (keine Bandendemarke nach der Programmdatei) oder ein von Null verschiedener Wert (Bandendemarke nach der Programmdatei) angegeben werden.

Wird ***Dateiname*** nicht vergeben (nur bei Kassettenspeicherung erlaubt), so wird überhaupt kein Name gespeichert. Wird für ***Dateiname*** eine Zeichenkettenvariable gesetzt, so muß diese in Klammern angegeben werden.

Beispiele:

```
SAVE
SAVE"TESTPRG"
SAVE(A$),1,0
```

Weitere Beispiele sowie eine detaillierte Beschreibung der SAVE–Anweisung bei Floppy–Disk–Betrieb finden sich in Kapitel 6.2.

SGN–Funktion

Format: v= **SGN(x)**

Zweck: Liefert das Vorzeichen des Argumentes x in folgender codierter Form:

$x > 0$ liefert 1

$x = 0$ liefert 0

$x < 0$ liefert –1

Beispiel:

```
ON SGN(A)+2 GOTO 100,200,300
```

Das Programmm verzweigt nach Zeile 100, wenn der Wert von A negativ ist, nach Zeile 200, wenn er Null ist und nach Zeile 300, wenn er positiv ist.

SIN–Funktion

Format: **v= SIN(x)**

Zweck: Liefert den Sinus von ***x*** im Bogenmaß. Die Berechnung von SIN(x) erfolgt binär in Gleitkommadarstellung. Zwischen SIN(x) und COS(x) besteht der Zusammenhang

COS(x) = SIN(x + 3.14159265/2)

Beispiel:

```
PRINT SIN(1.5)
.997494987
READY.
```

SPC–Funktion

Format: **SPC(*n*)**

Zweck: Liefert ***n*** Leerstellen. SPC kann nur in Verbindung mit der PRINT– oder PRINT#–Anweisung verwendet werden. ***n*** muß Werte zwischen 0 und 255 einnehmen.

Beispiel:

```
PRINT"HIER"SPC(15)"DA"
HIER               DA
READY.
```

SQR–Funktion

Format: ***v*=SQR(*x*)**

Zweck: Liefert die Quadratwurzel von ***x***. ***x*** muß größer oder gleich Null sein.

Beispiel:

```
10 FOR X=10 TO 25 STEP 5
20 PRINT X,SQR(X)
30 NEXT
RUN
10                 3.16227766
15                 3.87298335
20                 4.47213595
25                 5
READY.
```

STATUS–Systemvariable

Format: ***v*=STATUS**
***v*=ST**

Wirkung: Liefert ein Rechnerstatusbyte, dessen Inhalt auf Grund der letzten Ein/Ausgabe-Operation gesetzt wird. Dabei gelten je nach benutztem Ein–/Ausgabegerät die folgenden Statuswerte:

ST–Bit	ST–Wert	Kassette	ser. IEC–Bus
0	1		Zeitablauf beim Schreiben
1	2		Zeitablauf beim Lesen
2	4	kurzer Block	
3	8	langer Block	
4	16	fataler Fehler	
5	32	Prüfsum–menfehler	
6	64	Dateiende	Datenende
7	–128	Bandende	Gerät nicht angeschlossen

Beispiel:

```
10 OPEN6,1,2,"MASTERFILE"
20 GET#6,A$
30 IF ST AND 64 THEN 60
40 ?A$
50 GOTO 20
60 ?A$:CLOSE6
```

Die Datei MASTERFILE wird bis zum Dateiende von Kassette gelesen und angezeigt.

STOP–Anweisung

Format: **STOP**

Zweck: Bricht ein laufendes Programm ab und setzt den Interpreter in den Direkt–Modus.

Bemerkungen: STOP–Anweisungen dürfen an beliebiger Stelle in einem Programm stehen. Wird ein STOP ausgeführt, so meldet der Interpreter dies mit:

BREAK IN *nnnnn*

wobei ***nnnnn*** die Zeilennummer ist, bei der das Programm abgebrochen wurde.

Die STOP–Anweisung schließt nicht, wie die END–Anweisung ggfs. eröffnete Dateien. Nach einer STOP–Anweisung kann das Programm durch Eingabe von CONT (s. dort) im Direkt–Modus fortgesetzt werden.

Beispiel:

```
10 INPUT A,B,C
20 K=(A+3)/2:L=B*3
30 STOP
40 M=C*K+100:PRINT M
RUN
? 1,2,3
BREAK IN 30
READY.
CONT
106
READY.
```

STR$–Funktion

Format: ***v$* = STR$(*x*)**

Zweck: Liefert die Zeichenkettendarstellung von ***x***.

Beispiel:

```
10 INPUT"GIB BITTE EINE ZAHL EIN";N
20 PRINT N,LEN(STR$(N))
30 GOTO 10
RUN
GIB BITTE EINE ZAHL EIN? -124
-124      4
GIB BITTE EINE ZAHL EIN? 2
2         2
GIB BITTE EINE ZAHL EIN? usw.
```

Im zweiten Fall ist die Länge von STR$(2) deshalb 2, weil in der Zeichenkettendarstellung von positiven Zahlen der Zahl immer eine Leerstelle vorangestellt wird.

SYS–Anweisung

Format: **SYS *Ausdruck*[*,Parameterliste*]**

Zweck: übergibt die Programmsteuerung an ein Unterprogramm in Maschinensprache, das bei einer spezifizierten Adresse beginnt (s.a. USR–Funktion).

Bemerkungen: Der Wert von ***Ausdruck*** muß eine ganze Zahl zwischen 0 und 65535 sein. Er bezeichnet die Adresse im Programmspeicher des C128, bei der das Unterprogramm in Maschinensprache beginnt. Die Rückkehr in das BASIC-Hauptprogramm erfolgt durch den Assemblerbefehl RTS.

In ***Parameterliste*** können Parameter angegeben werden, die dem Maschinensprache–Unterprogramm übergeben werden sollen.

Die Auswertung dieser Parameter muß jedoch vom Maschinensprache–Unterprogramm vorgenommen werden!

Beispiel:

```
SYS 7*2↑12,X,Y
```

Die Zeichenfolge **,X,Y** muß vom Maschinenspracheunterprogramm, das bei der Adresse 28676 beginnt, so ausgewertet werden, daß der Textzeiger des Interpreters nach Beendigung des Unterprogramms auf das **Y** zeigt.

TAB–Funktion

Format: **TAB(*n*)**

Zweck: Tabuliert über ***n*** Spalten in der aktuellen Bildschirmzeile. Steht der Cursor vor der Ausführung von TAB(*n*) bereits rechts von der *n*–ten Spalte, so werden ***n*** Spalten der Folgezeile übertabuliert.

TAB bezieht sich immer auf den Zeilenanfang in der äußerst linken Bildschirmspalte (Spalte 0). Die äußerst rechte Position in einer Zeile ist dann Spalte 39 (wahlweise Spalte 79 im C128–Modus).

n muß zwischen Null und 255 liegen.

TAB kann nur in Verbindung mit der PRINT– oder PRINT#–Anweisung verwendet werden.

Beispiel:

```
10 PRINT"WARE"TAB(15)"BETRAG":PRINT
20 READ A$,B$
30 PRINT A$TAB(15)B$
40 DATA "BUTTER","DM 2.50"
RUN
WARE           BETRAG

BUTTER         DM 2.50
READY.
```

TAN–Funktion

Format: ***v*=TAN(*x*)**

Zweck: Liefert den Tangens von ***x*** im Bogenmaß. Die Berechnung von TAN(*x*) erfolgt binär im Gleitkommaformat.

Beispiel:

```
PRINTTAN(5)/2
-1.6902575
READY.
```

TIME–Systemvariable

Format: **v=TIME**
v=TI

Zweck: Liefert den momentanen Stand der internen System-Uhr, die alle 1/60 Sekunde fortgeschrieben wird. Die System-Uhr ist keine Echtzeit–Uhr. Sie wird beim Einschalten des Rechners initialisiert.

Beispiel:

```
PRINTTI
154788
READY.
```

TIME$–Anweisung
TIME$–Systemvariable

Format:

TIME$ = *x$* als Anweisung
TI$ = *x$*
***v$* = TIME$** als Systemvariable
***v$* = TI$**

Zweck:

Als Anweisung wird die Systemuhr auf eine gewünschte Zeit in Form einer 6–Byte–Zeichenkette in der Anordnung:

hhmmss (Stunden Minuten Sekunden)

gesetzt. Diese Uhrzeit wird vom System zeitgerecht fortgeschrieben und kann jederzeit als Systemvariable ausgelesen werden.

Beispiel:

```
10 INPUT"BITTE ZEIT (HHMMSS)
         EINGEBEN";TI$
20 FOR I=1 TO 1000:NEXT
30 PRINT TI$
RUN
BITTE ZEIT (HHMMSS) EINGEBEN? 141223
141234
READY.
```

USR–Funktion

Format: ***v*= USR(*x*)**

Zweck: Verzweigt zu einem Maschinensprache–Unterprogramm, dessen Startadresse vorher in die Zellen mit der Adresse 85 und 86 der zero page (Organisationsspeicher des Interpreters, s. Anhang H) gespeichert werden muß. Zelle 85 enthält den niederwertigen und Zelle 86 den höherwertigen Adreßteil. Das Argument ***x*** wird im ersten Gleitkommaakkumulator des Interpreters übergeben, in den auch das Ergebnis des Unterprogramms abgelegt werden muß, damit das BASIC–Hauptprogramm dieses Ergebnis unmittelbar einer Variablen zuweisen kann.

Beispiel:

```
10 B=T*SIN(Y)
20 C=USR(B/2)
30 D=USR(B/3)
```

VAL–Funktion

Format: ***v*=VAL(*x$*)**

Zweck: Liefert den numerischen Wert einer Zeichenkette, die aus Ziffern besteht. Außerdem sind die Zeichen . + – und E an den richtigen Stelle erlaubt. Beginnt die Zeichenkette mit einem anderen Zeichen als einer Ziffer, einem Punkt, Plus– oder Minuszeichen, so liefert VAL(*x$*) Null.

Beispiel:

```
10 X$=".0053"
20 PRINT VAL(X$)
RUN
5.3E-03
READY.
```

VERIFY–Befehl

Format: **VERIFY** ["***Dateiname***"[,***Geräteadr***]]
[,***Verschieb***]

Zweck: Vergleicht ein gegenwärtig im Programmspeicher befindliches Programm mit einem auf einem spezifizierten Ausgabegerät gespeicherten Programm byteweise und meldet ggfs. Unterschiede.

Bemerkungen: ***Geräteadr*** ist mit 1 (Kassettenstation) voreingestellt.

Wird ***Dateiname*** nicht angegeben (nur bei Kassettenfiles erlaubt), so erfolgt der Vergleich mit dem ersten auf Kassette gefundenen Programm.

Verschieb Ein ganzzahliger Wert von 0 oder 1, der angibt, ob das zu vergleichende Programm am Anfang des BASIC–Programm–Speichers (0) oder bei einer anderen Adresse (1) (z.B. Maschinenspracheunterprogramm) beginnt. Voreingestellt ist hier der Wert 0.

Beispiele:

```
VERIFY "PRGFILE"
PRESS PLAY ON TAPE
OK
FOUND PRGFILE
VERIFYING
VERIFY OK oder VERIFYING ERROR
READY.
```

VERIFY "MULT",8,1

Das Maschinensprache–Programm MULT auf Diskette wird mit dem speicherresidenten Programm verglichen, das nicht am Anfang des BASIC–Programmspeichers beginnt.

WAIT–Anweisung

Format: **WAIT *Adresse,n*[,*m*]**

Zweck: Hält die Programmausführung an, bis eine angegebene Speicherzelle des C128 ein spezifiziertes Bitmuster angenommen hat.

Bemerkungen: Zur Prüfung des spezifizierten Bitmusters wird zwischen dem Inhalt der durch ***Adresse*** gekennzeichneten Speicherzelle und dem Ganzzahl–Ausdruck ***m*** eine Exklusiv–ODER–Verknüpfung gebildet. Dieses Ergebnis wird durch ein logisches UND mit dem Ganzzahl–Ausdruck ***n*** verknüpft. Wenn das Ergebnis Null ist, wird das Bitmuster der spezifizierten Speicherzelle erneut gegestet. Erst wenn das Ergebnis von Null verschieden ist, wird die nächste BASIC–Anweisung ausgeführt. Wenn ***m*** weggelassen wird, wird sein Wert mit Null angenommen.

> **Achtung**
> Die WAIT–Anweisung kann nicht mit der STOP–Taste abgebrochen werden.

Beispiele:

```
WAIT 1,16,16
```

Die Programmausführung wird solange angehalten, bis eine Taste der Kassettenstation gedrückt wird.

```
WAIT A,2↑n
```

mit ***n***=0,1,. . . .,7

Die Programmausführung wird solange angehalten bis das Bit ***n*** der durch A spezifizierten Speicherzelle logisch 1 ist.

5.3 Farben und Grafik im C64–Modus

Wir haben nun schon den BASIC–Sprachumfang des C64–Modus kennengelernt. Eine der hervorragenden Eigenschaften dieses Modus besteht jedoch auch in der Möglichkeit, Farbgrafiken zu erstellen. Ein einfaches Beispiele ist das im Laufe dieses Abschnittes beschriebene Springballprogramm. Aber die Möglichkeiten des C64–Modus sind unvergleichlich größer.

In diesem Abschnitt werden wir Ihnen zeigen, wie Sie Farbgrafiken erstellen und z.B. in Spielen einsetzen können.

Wir gehen dabei von den Standardfarben des Bildschirms (hellblaue Schrift auf dunkelblauem Hintergrund mit hellblauem Rand) aus und wollen Ihnen zeigen, wie Sie die Farben ändern können und wie Sie die vielen Grafiksymbole im C64–Modus einsetzen können.

Farbgebung mit PRINT–Befehlen

Die Einstellung der Farben geschieht anhand eines Musters, das sich mit einfachen Mitteln auf dem Bildschirm ihres Fernsehgeräts oder Monitors erzeugen läßt.

Sie benutzen dazu die CTRL–Taste auf der linken Seite der Tastatur. Die CTRL–Taste wird stets zusammen mit einer anderen Taste so verwendet, daß zunächst die CTRL–Taste niedergehalten und dann die gewünschte andere Taste gedrückt wird.

Sie haben eine Skala von 16 Zeichenfarben zur Verfügung. Durch Benutzung der CTRL–Taste und der Zifferntasten können Sie folgende Farbkombinationen erzeugen:

1	2	3	4
schwarz	weiß	rot	grün
5	6	7	8
violett	dunkelgrün	blau	gelb

Durch Verwendung der Commodore–Taste ganz unten links auf der Tastatur zusammen mit den Zifferntasten erhalten Sie folgende Farben:

1	2	3	4
hellbraun	braun	hellrot	dunkelgrau
5	6	7	8
grau	hellgrün	hellblau	hellgrau

Geben Sie nun NEW ein und machen folgendes Experiment:

Nach dem Zeilenanfang 10 PRINT" drücken Sie zusammen die CTRL– und die 1–Taste. Lassen Sie dann die CTRL–Taste los und drücken die S–Taste. Betätigen Sie nun die CTRL–Taste zusammen mit der 2–Taste und dann die P–Taste allein usw.
Wählen Sie auf diese Weise eine Farbe nach der anderen und geben Sie zwischen den Buchstaben das Wort Spektrum ein.

```
10PRINT"  S P E K T R U M"
          ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
CRTL      1 2 3 4 5 6 7 8
```

Wie bei der Cursorsteuerung werden auch die Farbsteuerzeichen als grafische Zeichen dargestellt.

Bei gleichzeitigem Drücken von CTRL– und 3–Taste erscheint das englische Pfund–Zeichen und von CTRL– und 7–Taste der Pfeil nach links.

Die nachfolgende Tabelle gibt eine Zusammenstellung dieser Farbcodes.

TASTE	FARBE	AUSGABE	TASTE	FARBE	AUSGABE
CTRL 1	SCHWARZ		C= 1	ORANGE	
CTRL 2	WEISS		C= 2	BRAUN	
CTRL 3	ROT		C= 3	HELLROT	
CTRL 4	TÜRKIS		C= 4	GRAU 1	
CTRL 5	VIOLETT		C= 5	GRAU 2	
CTRL 6	GRÜN		C= 6	HELLGRÜN	
CTRL 7	BLAU		C= 7	HELLBLAU	
CTRL 8	GELB		C= 8	GRAU 3	

Wie Sie schon bemerkt haben, werden die Steuerzeichen nur beim Listen des Programms sichtbar, bei der Programmausführung, d.h. beim Drucken des Wortes Spektrum tauchen sie nicht mehr auf. Der Text wird vielmehr (mit den entsprechenden Farbvariationen) ausgegeben.

Spielen Sie nun einige Möglichkeiten durch, um mit der Farbsteuerung vertraut zu werden und vergessen Sie nicht, daß Sie noch weitere Farbmöglichkeiten im Zusammenhang mit der Commodore–Taste haben.

- Anmerkung: Nach Beendigung eines Programms, bei dem Sie die Farbsteuerung angewandt haben, bleibt der C128 in dem Modus, den Sie zuletzt eingeschaltet haben. Durch Drücken der Tasten RUN/STOP und RESTORE kommen Sie in den Normalfarbmodus zurück.

Farb–CHR$–Codes

Einige Farben können über die CHR$–Funktion direkt angesprochen werden. Sie haben dieselbe Wirkung wie das Drücken der CTRL–Taste und der entsprechenden Ziffern– Taste.

Probieren Sie folgendes Beispiel aus:

```
10 PRINT CHR$(147):REM CLR
20 PRINT CHR$(30);"CHR$(30) FAERBT MICH"
```

Die Schrift sollte nun grün sein. In vielen Fällen ist die Verwendung der CHR$–Funktion zur Farbsteuerung wesentlich einfacher als die Verwendung der Farbtasten.

Das nachfolgende Programm erzeugt Farbbalken auf dem Bildschirm.

```
10 REM AUTOMATISCHE FARBBALKEN
20 PRINT CHR$(147):REM CHR$(147)= CLR
30 PRINTCHR$(18);"    ";:REM REVERSER BALKEN
40 CL= INT(16*RDN(1))+1
50 ON CL GOTO 60,70,80,90,100,110,120,130,140,
       150,160,170,180,190,200,210
60 PRINTCHR$(5);:GOTO30
70 PRINTCHR$(28);:GOTO30
80 PRINTCHR$(30);:GOTO30
90 PRINTCHR$(31);:GOTO30
100 PRINTCHR$(144);:GOTO30
110 PRINTCHR$(156);:GOTO30
120 PRINTCHR$(158);:GOTO30
130 PRINTCHR$(159);:GOTO30
140 PRINTCHR$(129);:GOTO30
150 PRINTCHR$(149);:GOTO30
160 PRINTCHR$(150);:GOTO30
170 PRINTCHR$(151);:GOTO30
180 PRINTCHR$(152);:GOTO30
190 PRINTCHR$(153);:GOTO30
200 PRINTCHR$(154);:GOTO30
210 PRINTCHR$(155);:GOTO30
```

Farbgebung durch PEEK und POKE

Nun lernen Sie eine Methode kennen, wie Sie sich im C64–Modus umschauen und Informationen an die von Ihnen ausgewählte Stelle bringen können, um den Computer zu steuern.

Die Inhalte einiger Speicherzellen haben eine ganz bestimmte Bedeutung. So gibt es Speicherzellen, in denen im C64–Modus festgelegt wird, welche Bildschirm– oder Rahmenfarbe ausgewählt ist, welches Zeichen auf dem Bildschirm in welcher Farbe angezeigt wird, und wo es dort stehen soll.

Wenn man den Inhalt dieser Speicherplätze ändert, so kann man damit auch die o.a. Parameter ändern.

Farben können geändert werden. Objekte können auf dem Bildschirm erscheinen und sich bewegen.

Schreibt man Werte direkt in den Computer–Speicher, so geschieht das mit Hilfe des POKE–Befehls (s. dort in Kapitel 5.2). Man sagt deshalb auch, daß man einen Wert in eine Speicherzelle **gepoked** hat.

Man kann Speicherstellen durch ihre Adresse ansprechen:

z.B. 53280 oder 53281

Wir haben hier zwei Speicherplätze angegeben, deren Inhalte die Bildschirm– und die Hintergrundfarbe bestimmen.

Geben Sie bitte folgendes ein:

```
POKE53281,7
```

Nach dem Drücken der RETURN–Taste erhalten Sie einen gelben Bildschirm, da der Wert 7 die Farbe gelb definiert und wir sie in die Speicherzelle, die die Bildschirmhintergrundfarbe wählt, **gepoked** haben. Versuchen Sie dasselbe mit anderen Zahlenwerten.

Sie können jede Zahl zwischen 0 und 255 verwenden; sinnvoll sind jedoch nur Zahlen zwischen 0 und 15.

Der nachstehenden Tabelle können Sie entnehmen, wie die Zahlenwerte den Farben zugeordnet sind.

0	schwarz	8	hellbraun
1	weiß	9	braun
2	rot	10	rosa
3	grün	11	dunkelgrau
4	violett	12	grau
5	dunkelgrün	13	hellgrün
6	blau	14	hellblau
7	gelb	15	hellgrau

Diese Farben können bei verschiedenen Farbmonitoren durchaus variieren.

Nun wollen wir uns verschiedene Kombinationen von Hintergrund– und Rahmenfarben anschauen.

Dabei hilft uns folgendes Programm:

```
10 FOR BA = 0 TO 15
20 FOR BO = 15 TO 1 STEP -1
30 POKE 53280, BA
40 POKE 53281, BO
50 FOR X = 1 TO 2000 : NEXT X
60 NEXT BO : NEXT BA
```

Zwei Schleifen werden ineinander geschachtelt, um alle Kombinationen zu erfassen. Die zusätzliche Schleife in Zeile 50 verzögert den ganzen Vorgang nur ein bißchen.

Wenn Sie die Werte, die für den jeweiligen Farbwechsel verantwortlich sind, während des Farbwechselprogramms auf dem Bildschirm sichtbar machen wollen, fügen Sie folgende Programmzeile hinzu:

```
25 PRINT CHR$(147);"RAHMEN=";PEEK(53280)AND15,
      "HINTERGRUND=";PEEK(53281)AND15
```

Bildschirmgrafik

Bisher haben Sie den Cursor nur durch PRINT–Befehle gesteuert.

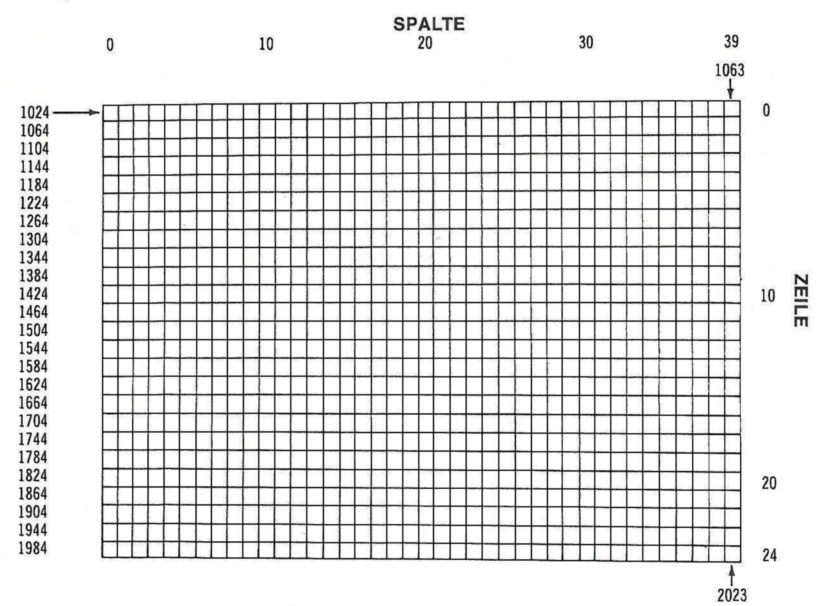
Damit läßt sich jeder Punkt des Bildschirms erreichen; diese Methode ist jedoch im allgemeinen sehr langsam und verbraucht wertvollen Speicherplatz.

Im C64–Modus gibt es aber auch Speicherplätze, die die Bildschirmfarben bestimmen.

Der Bildschirmspeicher

Der Bildschirm im C64–Modus besteht aus 25 Zeilen zu je 40 Zeichen. Daher können Sie 1000 Zeichen auf ihm unterbringen und brauchen natürlich genausoviele Speicherplätze, die Informationen darüber enthalten, welche Zeichen sich in den einzelnen Bildschirmenpositionen befinden. Sie können sich den Bildschirm als rechtwinkliges Gitter vorstellen (s. nächste Seite), wobei jedes kleine Rechteck in diesem Gitter einer Speicherzelle entspricht.

Sie können in diese Speicherzellen Werte zwischen 0 und 255 hineinschreiben (**poken**).



Der Bildschirmspeicher im C64–Modus beginnt bei der Adresse 1024 und reicht bis zur Adresse 2023.

Die Speicherzelle 1024 entspricht dabei der linken oberen Ecke des Bildschirms und die Speicherzelle 2023 der unteren rechten Ecke.

Um eine bestimmte Position auf dem Bildschirm zu erreichen, können Sie nach folgendem Rechenschema vorgehen:

Bildschirmadresse = 1024 + Spalte + 40 * Zeile

Nehmen wir nun an, Sie wollen einen Ball etwa in der Mitte des Bildschirms erscheinen lassen (Spalte 20, Zeile 12). Die Speicheradresse dafür errechnet sich wie folgt:

1024 + 20 + 40 * 12 = 1524

Nun löschen Sie den Bildschirm durch gleichzeitiges Drücken der SHIFT– und der CLR/HOME–Tasten und geben Sie dann ein:

```
POKE 1524,81
```

Dabei bedeuten die Zahlen folgendes:

1524 = Adresse der Bildschirmmitte
81 = Zeichencode (Ball)

Der Farbspeicher

Sie haben nun einen Ball in der Mitte des Bildschirms erzeugt. Dies haben Sie erreicht, ohne den PRINT–Befehl zu nutzen. Sie haben einen Wert direkt in den Bildschirmspeicher geschrieben. Leider können Sie den Ball noch nicht sehen, er hat nämlich dieselbe Farbe wie der Hintergrund.

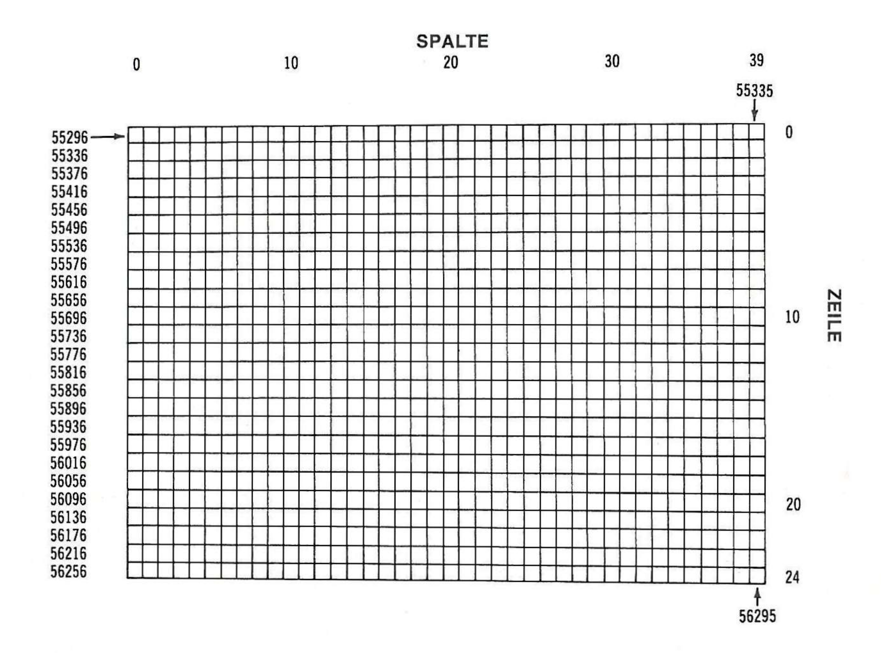
Es gibt jedoch einen weiteren Speicher im C64–Modus, bei dem Sie durch Ändern der Speicherinhalte die Farben von einzelnen Zeichen auf dem Bildschirm bestimmen können. Geben Sie folgendes ein:

```
POKE 55796,2
```

Dabei bedeuten die einzelnen Zahlen folgendes:

55796 = Adresse der Farbzelle für die Bildschirmmitte
2 = Farbcode (rot)

Der Ball wird daraufhin rot. Da Sie außer der Information über das Zeichen in einer bestimmten Bildschirmposition auch die Farbinformationen brauchen, gehörten zu jeder Position auf dem Bildschirm zwei Speicherstellen. Der Farbspeicher beginnt bei der Adresse 55296 (linke obere Ecke) und hat natürlich, wie der Bildschirmspeicher, 1000 Speicherzellen.



Die Farbcodes liegen zwischen 0 und 15 und und entsprechen denen, die wir oben benutzt haben, um Hintergrund und Rahmen zu ändern (siehe Farbtabelle weiter oben).

Die Formel zur Berechnung der Farbspeicheradresse entspricht jener zur Berechnung der Bildschirmadresse:

Farbspeicheradresse = 55296 + Spalte + 40 * Zeile

Ein Springballspiel

```
10 PRINTCHR$(147):REM CLR/HOME
20 POKE53280,7:POKE53281,0
30 X=1:Y=1
40 DX=1:DY=1
50 POKE 1024+X+39*Y,81
55 POKE 55296+X+39*Y,1
60 FORT=0TO10:NEXT
70 POKE 1024+X+39*Y,32
80 X=X+DX
90 IF X=0 OR X=39 THEN DX=-DX
100 Y=Y+DY
110 IF Y=0 OR Y=24 THEN DY=-DY
120 GOTO 50
```

Nachdem in Zeile 10 der Bildschirm gelöscht wurde, wird in Zeile 20 die Hintergrundfarbe schwarz und die Rahmenfarbe gelb gewählt.

Die Variablen X und Y in Zeile 30 stehen für die Zeile und Spalte, in der sich der Ball augenblicklich befindet.

Die Variablen DX und DY der Zeile 40 geben die horizontale und vertikale Bewegungsrichtung des Balls an. DX= +1 entspricht einer Bewegung nach rechts. DX= –1 entspricht einer Bewegung nach links. Analog dazu entsprechen DX und DY einer Bewegung nach oben bzw. unten.

In Zeile 50 wird der Ball in der durch Zeilen– und Spaltennummer bestimmten Position angezeigt.

In Zeile 60 ist eine Verzögerungsschleife eingefügt.

In Zeile 70 wird der Ball durch Überschreiben mit einem Leerzeichen gelöscht.

In Zeile 80 wird durch Addition von DX der Ball in der richtigen Richtung bewegt.

Das Vorzeichen von DX wird umgedreht, wenn in Zeile 90 festgestellt wird, daß der Ball den linken oder rechten Rand berührt.

In den Zeilen 100 und 110 geschieht dasselbe für den oberen und unteren Rand.

Die Zeile 120 bewirkt einen Sprung in die Zeile 50, wo der Ball in die neu berechnete Position auf dem Bildschirm ausgegeben wird.

Wenn Sie in Zeile 50 die 81 gegen eine andere Zahl austauschen, so können Sie den Ball durch ein beliebiges Zeichen ersetzen.

Durch folgende Ergänzung können wir das Programm noch ein bißchen intelligenter machen:

```
21 FORL=1TO 10
25 POKE1024+INT(RND(1)*1000),166
27 NEXT L
115 IF PEEK(1024+X+40*Y)=166 THEN DX= -DX:GOTO 80
```

Die Zeilen 21 – 27 besetzen zufällig gewählte Bildschirmpositionen mit Hindernissen.

In Zeile 115 wird mit Hilfe der PEEK–Funktion geprüft, ob der Ball gegen ein Hindernis stößt. Ist dies der Fall, wird die Bewegungsrichtung geändert.

5.3.1 Sprite–Grafik im C64–Modus

Einleitung

In den vorhergehenden Kapiteln haben Sie gesehen, wie man mit dem PRINT–Befehl den Bildschirm als Tabelle formatieren kann und wie man Hilfe des POKE–Befehls an beliebigen Stellen des Bildschirm Zeichen ausdrucken kann.
Die Konstruktion von bewegten Bildern verursacht mit beiden Methoden einige Schwierigkeiten, da die Objekte aus vorgefertigten Symbolen zusammengesetzt werden müssen.
Weiterhin bringt das Bewegen und Kontrollieren dieser Objekte einen großen Aufwand an Befehlen mit sich. Durch die begrenzte Anzahl von Grafiksymbolen sind Sie in der Formgebung der Objekte stark eingeschränkt.

Durch die Verwendung von **Sprites** entfallen die meisten oben aufgeführten Probleme. Ein SPRITE stellt ein frei programmiertes Objekt in hochauflösender Grafik dar, dem durch BASIC–Befehle jede beliebige Form gegeben werden kann. Durch einfache Angabe der Position kann das SPRITE auf dem Bildschirm verschoben werden. Die nötigen Berechnungen werden im C64–Modus intern erledigt.

Aber Sprites haben noch mehr Vorteile. Ihre Farbe kann geändert werden, der Zusammenstoß zweier Sprites kann auf einfache Weise registriert werden. Ein Sprite kann sich vor oder hinter einem anderen vorbeibewegen und man kann mit einem Befehl seine Größe ändern.

All diesen Vorteilen stehen nur geringe Schwierigkeiten beim Programmieren gegenüber. Zugegeben, Sie müssen noch einiges darüber lernen, wie man im C64–Modus arbeitet und wie die Zahlen intern verarbeitet werden. Aber schließlich ist es doch ganz interessant und auch gar nicht so schwierig.

Wenn Sie die im folgenden angegebenen Beispiele sorgfältig durcharbeiten, werden Sie bald mit selbstentworfenen Sprites einige Kunststücke anstellen können.

Der Entwurf von Sprites

Die Sprites werden von einem speziellen Grafikbaustein, dem VIC (Video Interface Chip), unterstützt. Die Arbeit, die das Entwerfen der Sprites, das Kontrollieren ihrer Bewegungen und Positionen und die Farbgebung macht, wird zum größten Teil von diesem Chip übernommen.

Der Bereich, in dem die Sprites generiert werden, besteht aus 64 Speicherstellen der Art, wie Sie sie schon bei der Behandlung des Bildschirm– und Farbspeichers kennengelernt haben.
Jede dieser Speicherstellen kann man sich in 8 kleine Teile; sog. Bits, unterteilt denken, die einzeln an– und ausgeschaltet werden können und auf diese Weise die Form der Sprites bestimmen. In welchem Zustand (an oder aus) die einzelnen Bits sind, wird durch die Zahl bestimmt, die in das betreffende Register geschrieben wird.

Zusätzlich zu diesen speziellen Registern werden wir auch Speicher im C64–Modus nutzen, um Informationen über die Form der Sprites zu speichern.
In 8 Speicherzellen (direkt hinter dem Bildschirmspeicher) werden Daten abgelegt, die dem Computer mitteilen, in welchem Speicherbereich die Daten für die Sprites gespeichert sind.

Wie sind die Sprites nun aufgebaut?

Wie Sie wissen, besteht der Bildschirm aus 25 Zeilen zu je 40 Zeichen. Jede der sich daraus ergebenden 1000 Bildschirmpositionen können Sie durch

POKE–Befehle mit einem Zeichen belegen. Dieses Zeichen selbst ist wiederum aus einer 8 x 8–Punktmatrix zusammengesetzt.

Beim Entwerfen eines Sprites können Sie nun jeden Punkt dieser Matrix einzeln ansprechen und erhalten dadurch eine Auflösung von 320 x 200 (horizontal x vertikal) Punkten für den gesamten Bildschirm.

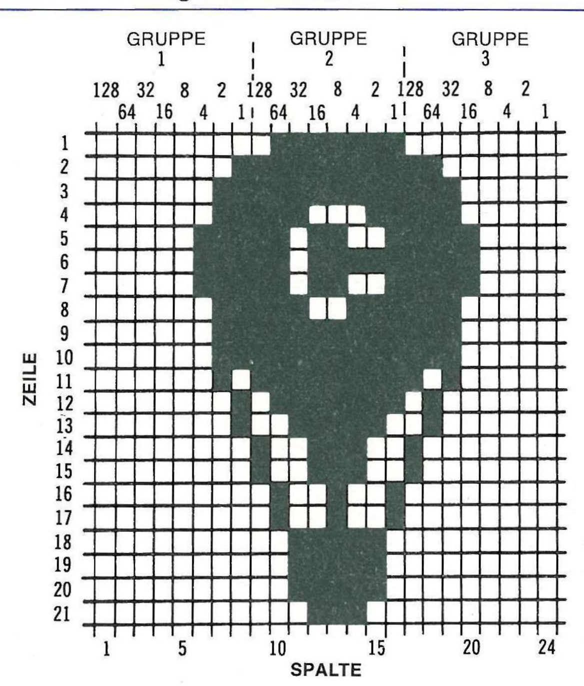
Ein Objekt, das Sie auf diese Weise zusammensetzen, darf maximal 24 Punkte breit und 21 Punkte hoch sein.

Als Beispiel haben wir aus diesem 24 x 21–Punktefeld einen Ballon konstruiert, der unten abgebildet ist.

Am besten entwerfen Sie das Objekt auf fein gerastertem Papier (z.B. Millimeterpapier), auf dem Sie ein 24 Kästchen breites und 21 Kästchen hohes Feld markieren.
Zeichnen Sie zuerst die Form so ein, wie Sie Ihnen vorschwebt, und füllen Sie dann die Felder aus, die von den von Ihnen gezogenen Linien geschnitten werden.
Auf diese Weise haben Sie die Form des Sprites festgelegt. Sie müssen diese aber nun noch in Daten umwandeln, die der Computer verarbeiten kann.

Schreiben Sie zu diesem Zweck an den oberen Rand des 24 x 21–Punktefeldes dreimal hintereinander die Zahlenreihe 128, 64, 32, 16, 8, 4, 2, 1.
Die Zeilen des Feldes nummerieren Sie von 1 – 21 durch.

Legen Sie sich nun für jede Zeile drei Wertetabellen an, wie sie nachfolgend aufgeführt sind.
Tabelle 1 entspricht den ersten 8 Positionen einer Zeile, die Tabellen 2 und 3 entsprechen den Positionen 9 – 16 bzw. 17 – 24.
Über die ganzen Felder schreiben Sie nun wieder die oben angeführte Zahlenreihe von 128 – 1. Wir ordnen nun in unserer Spritezeichnung jedem ausgefüllten Feld die Zahl 1 und jedem leeren Feld die Zahl 0 zu.
Dann legen wir für jede Zeile drei Tabellen an und schreiben die entsprechenden Werte in die einzelnen Felder.



Nehmen wir als Beispiel die Zeile 1 des oben dargestellten Sprites. Die ersten 8 Felder sind leer, d.h. unsere Tabelle 1 enthält nur Nullen. Um daraus eine Zahl zu errechnen, die der Computer verarbeiten kann, müssen wir den Inhalt mit dem Wert multiplizieren, der über diesem Feld steht und die Produkte addieren. Da alle Felder 0 enthalten, erhalten wir als erstes Datum als Summe eine Null.

Die zweite Tabelle der Reihe ist nachstehend abgebildet.

128	64	32	16	8	4	2	1
0	1	1	1	1	1	1	1
↑	↑	↑	↑	↑	↑	↑	↑

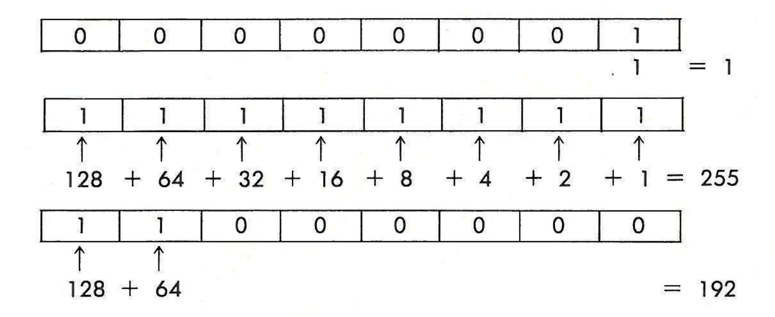
0 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127

Die dritte Tabelle der ersten Reihe enthält wieder Nullen. Die Summe ist also auch 0. Die 1. Reihe unseres Sprites wird also durch die Zahlen 0,127,0 beschrieben.

Um sie leicht in einem Programm verwerten zu können, schreiben wir sie als DATA–Zeile:

```
DATA 0, 127, 0
```

Als weiteres Beispiel die Daten der zweiten Reihe unseres Sprites:



Für die Reihe zwei erhalten wir also die DATA–Zeile:

```
DATA 1,255,192
```

Um mit der beschriebenen Methode vertraut zu werden, sollten Sie nun die restlichen 19 DATA–Zeilen selbst berechnen.

Nach dem Sie dies getan haben, probieren Sie folgendes Programm:

```
10 REM BALLONFAHRT
20 PRINTCHR$(143):V=53248:REM BASISADR. DES VIC
30 POKEV+21,4:REM SPRITE 2 AKTIVIEREN
40 POKE2042,13:REM DATEN FUER SPRITE 2
50 FORN=0 TO 62:READQ:POKE832+N,Q:NEXT
60 FORX=0 TO 200
70 POKEV+4,X:REM NEUE X-KOORDINATE
80 POKEV+5,X:REM NEUE Y-KOORDINATE
90 NEXTX:GOTO60
100 DATA 0,127,0,1,255,192,3,255,224,3,231,244
110 DATA 7,217,240,7,223,240,7,217,240,3,231,224
120 DATA 3,255,224,3,255,224,2,255,160,1,127,64
130 DATA 1,62,64,0,156,128,0,156,128,0,73,0,0,73,0,0
140 DATA 62,0,0,62,0,0,62,0,0,28,0
```

Wenn Sie alles richtig eingegeben haben, fliegt ein Ballon ruhig über den Bildschirm. Um das Programm zu verstehen, müssen Sie wissen, welche Sprite–Speicherstellen (Register) welche Funktionen kontrollieren.

Sie können das der folgenden Tabelle entnehmen:

Register	Beschreibung
0	X–Koordinate von Sprite 0
1	Y–Koordinate von Sprite 0
2–15	Bedeutung paarweise wie 0 und 1 für Sprites 1 – 7
16	Höchstes Bit–X–Koordinate
21	1 = Sprite aktiviert; 0 = Sprite inaktiiviert
23	Sprite wird in Y–Richtung vergrößert
29	Sprite wird in X–Richtung vergrößert
39–46	Farben der Sprites 0 – 7

Sie müssen außerdem wissen, in welchem 64er Block die Daten eines bestimmten Sprites abgespeichert sind.

Diese Daten stehen in den 8 Registern direkt hinter dem Bildschirmspeicher.

2040	41	42	43	44	45	46	2047
↑	↑	↑	↑	↑	↑	↑	↑
SPRITE 0	1	2	3	4	5	6	7

Nun wollen wir Schritt für Schritt durchgehen, wie wir unsere Sprites auf den Bildschirm bringen können.

- Poken Sie in die Speicherstelle 21 den richtigen Wert (s. nächste Seite), damit das von Ihnen gewählte Sprite auf dem Bildschirm erscheint.

- Lassen Sie den Sprite–Zeiger auf die Speicherstelle zeigen, von der die Daten des Sprites gelesen werden sollen.

- Schreiben Sie mit POKE die Daten in diese Speicherstelle.

- Konstruieren Sie mit einer Schleife die X– und Y–Koordinaten für die Bewegung des Sprites.

- Sie können zusätzlich die Farben des Sprites oder seine Größe (X– oder/und Y–Richtung) ändern. Die Parameter für die Größenänderungen stehen in den Registern 23 und 29.

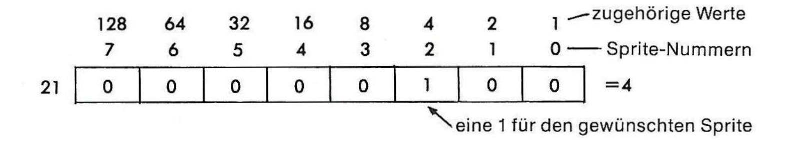
Einige Punkte des Programms wollen wir einmal näher erklären.

Zeile 20: `V = 53248`

Die erste Speicheradresse des VIC wird unter V abgespeichert. Zu diesem Wert brauchen nur noch die Nummern des Registers addiert werden, um die absolute Speicheradresse zu erreichen.

Zeile 30: `POKEV+21,4`

Dieser Befehl läßt Sprite 2 auf dem Bildschirm erscheinen. Wenn Sie die untenstehende Tabelle anschauen können Sie die Zugehörigkeit der einzelnen Sprites erkennen:



Jedes Sprite ist im Register 21 vertreten. So entspricht der Wert 4 dem Sprite 2. Sprite 7 entspicht dem Wert 128 und beide zusammen 132 (128 + 4). Wenn Sie die Sprites 2 und 7 einschalten wollten, müßten Sie Poke V + 21,132 eingeben.

Zeile 40: `POKE2042,13`

Der Computer wird angewiesen, die Daten für Sprite 2 (entspricht Speicherstelle 2042) aus Speicherblock 13 zu lesen (1 Speicherblock = 64 Bytes, d.h. 13*64 = 832).
Die zweite Zahl in der Poke–Anweisung ist die Anfangsadresse des zulesenden Sprites.

Ein Sprite verbraucht 63 Bytes. Der Inhalt einer Tabelle, die wir für die Ermittlung der Datazeilen zusammengestellt haben, entspricht einem Byte. Daher müssen also 63 Byte gelesen werden, um ein Sprite zu füllen.

Dies geschieht auf folgende Weise:

```
50 FOR N = 0 TO 62: READ Q: POKE832+N, Q: NEXT
```

Durch diese Schleife werden 63 Datenbytes in den 13. Block gelesen, der bei der Adresse 832 (13*64) beginnt.

```
60 FOR X = 0 TO 200
70 POKE V+4,X
80 POKE V+5,X
```

Da die Register 4 und 5 die x– und y–Koordinaten des Sprites 2 enthalten, bewirkt dieser Programmteil (natürlich zusammen mit NEXT), daß sich das Sprite 2 diagonal über den Bildschirm bewegt.

Da der Koordinatenursprung in der linken oberen Bildschirmecke liegt, verläuft die Bewegung von links oben nach rechts unten. Der Computer liest die Daten schnell genug, um die Bewegung kontinuierlich erscheinen zu lassen.

Wenn sich mehrere Sprites über dem Bildschirm bewegen sollen, wird jedem Objekt ein eigener Speicherbereich zugeordnet. Die Zeile 90 bewirkt den Rücksprung nach Zeile 60, wodurch der ganze Vorgang wiederholt wird.

Der Rest des Programms besteht aus DATA–Zeilen, die die Informationen über die Form des Ballons enthalten.

Fügen Sie nun folgende Zeile zu dem Programm hinzu und starten Sie es erneut:

```
55 POKE V+23,4:POKE V+29,4:REM SPRITE VERGROESSERN
```

Der Ballon ist jetzt in x– und y–Richtung doppelt so groß wie vorher. Der Grund dafür ist, daß die Zahl 4 in die Register 23 und 29 geschrieben wurden.

Es ist wichtig zu berücksichtigen, daß die linke obere Ecke des Sprites an ihrem Platz bleibt.

Um das Programm noch interessanter zu machen, fügen Sie folgende Zeilen hinzu:

```
30 POKE V+21,12
40 POKE 2042,13:POKE 2043,13
60 FOR X=1 TO 190
75 POKE V+6,X
85 POKE V+7,190-X
```

Ein weiteres Sprite (Nr. 3) ist auf dem Bildschirm erschienen, da wir in das Register 21 eine 12 gepoked haben. Die 12 schaltet die Sprites 2 und 3 ein.

Durch die Zeilen 75 und 85 wird das 3. Sprite auf dem Bildschirm bewegt. Die Speicherstellen V+6 und V+7 enthalten die x– und y–Koordinaten des Objekts.

Wenn Sie wollen, daß am Himmel noch ein bißchen mehr los ist, dann bringen Sie an Ihrem Programm noch die folgenden Ergänzungen an:

```
30 POKE V+21,28
40 POKE 2042,13:POKE 2043,13:POKE 2044,13
55 POKE V+23,12:POKE V+29,12
78 POKE V+8,X
88 POKE V+9,100
```

Durch den POKE–Befehl in Zeile 30 wird ein weiteres Sprite erzeugt.

In Zeile 40 wird festgelegt, daß alle drei Sprites ihre Daten dem 13. Block entnehmen.

In Zeile 55 werden die Sprites 2 und 3 in x– und y–Richtung auf das Doppelte vergrößert.

Zeile 78 bewirkt, daß sich Sprite 3 in Richtung der x–Achse bewegt.

Da in Zeile 88 für die y–Koordinate der Wert 100 fest vorgegeben wird, bewegt sich das 3. Sprite nur horizontal.

Farbgebung der Sprites

Aus der Tabelle für Zeichenfarben können Sie die gewünschte Farbkodierung (0 – 15) entnehmen.

Wenn Sie z.B. dem Sprite Nr. 1 die Farbe hellgrün geben wollen, müssen Sie folgenden Befehl eingeben:

```
POKE V+40,13
```

Sie werden vielleicht beim Ausprobieren des Beispielprogramms gemerkt haben, daß die Sprites nie den rechten Bildrand erreichen. Der Grund dafür ist, daß die Bildschirmbreite 320 Punkten entspricht, das X–Register jedoch nur Werte zwischen 0 und 255 enthalten kann. Wie kann man nun ein Objekt dazu bringen, sich über den ganzen Bildschirm zu bewegen?

Man benutzt zu diesem Zweck das Register Nr. 16, das das höchstwertige Bit (auch MSB = Most Sigfificant Bit) enthält. Das ist folgendermaßen zu verstehen:

Wird das n–te Bit dieses Registers gesetzt, so befindet sich das n–te Sprite in der Bildschirmposition, die einem X–Wert von mehr als 255 entspricht. Der aktuelle X–Wert ergibt sich dann, wenn man zu dem Wert, der im X–Register steht, 256 hinzuzählt.

Soll z.B. das Sprite Nr. 2 eine X–Position zwischen 256 und 320 einnehmen, so muß im Register Nr. 16 das zweite Bit gesetzt werden. Sie müssen also in dieses Register eine 4 poken.

```
POKE V+16,4
```

Nun zählen wir den Inhalt des X–Registers, das zum zweiten Sprite gehört (das ist Register Nr. 4) von 0 bis 63 hoch. Auf diese Weise erreichen Sie die restlichen X–Werte von 256 bis 319.

Sie begreifen das Konzept am besten, wenn Sie das folgende Programm analysieren, das eine leicht abgeänderte Version unseres bisherigen Sprite-Programms darstellt:

```
10 V=53248 : POKE V+21,4 : POKE 2042,13
20 FOR N=0 TO 62 : READ Q : POKE 832+N,Q : NEXT
25 POKE V+5,100
30 FOR X=0 TO 255
40 POKE V+4,X
50 NEXT
60 POKE V+16,4
70 FOR X=0 TO 63
80 POKE V+4,X
90 NEXT
95 POKE V+16,0
98 GOTO 30
```

Die Zeile 10 enthält die Anfangsadresse des VIC. Das Sprite 2 wird aktiviert und der Block von dem eingelesen wird, wird bestimmt.

In Zeile 20 wird aus den DATA–Zeilen in den Block 13 eingelesen.

In Zeile 25 wird die Y Koordinate festgelegt.

In Zeile 30 wird eine Schleife für die X Koordinaten 0 – 255 aufgebaut und in Zeile 40 ausgegeben.

In Zeile 60 wird das MSB für Sprite Nr. 2 gesetzt.

In Zeile 70 beginnt die Schleife, die das Sprite zum rechten Bildschirmrand wandern läßt.

Die Zeile 95 ist ebenfalls wichtig, da hier das MSB wieder ausschaltet wird, so daß die Bewegung wieder am linken Bildrand starten kann.

5.4 Musik im C64–Modus

Die Tonerzeugung im C64–Modus hat zwei Hauptanwendungsgebiete:

Das Spielen von Musikstücken und das Erzeugen von Klangeffekten.

Wir werden nur kurz darauf eingehen, wie ein Musikprogramm im allgemeinen aufgebaut ist und besprechen dann ein Musikprogramm, mit dem Sie experimentieren können.

Die Struktur eines Musikprogramms

Der Klang eines Tones wird durch vier Eigenschaften bestimmt:

- Tonhöhe
- Lautstärke
- Klangfarbe
- Anschlag

Die beiden letzten Eigenschaften bewirken, daß Sie überhaupt verschiedene Instrumente mit dem Gehör unterscheiden können. Gerade diese wichtigen Eigenschaften werden Sie daher mit Ihrem Programm beeinflussen wollen.

Im C64–Modus wird zu diesem Zweck ein elektronischer Baustein, der SID (Sound Interface Device) benutzt. Im SID sind eine Reihe von Speicherplätzen für die Parameter enthalten, die das gewünschte Klangbild zusammensetzen.

Sie wissen schon, daß im C64–Modus drei Stimmen gleichzeitig erzeugt werden können. Sehen wir uns zunächst die erste dieser Stimmen an. Die Basisadresse des SID wollen wir durch die Variable SI abkürzen:

SI = 54272

Die Tonhöhe wird physikalisch durch die Frequenz bestimmt. Die Frequenz wird im SID durch einen Parameter gespeichert, der Werte zwischen 0 und 65535 annehmen kann. Sie haben im vorherigen Abschnitt gelernt, daß man so große Zahlen nicht in einer Speicherzelle abspeichern kann. Wir müssen daher den Frequenzparameter in ein höherwertiges und ein niederwertiges Byte zerlegen.

Das höherwertige Byte wird Hi–Byte genannt und das niederwertige Lo–B-yte. Diese beiden Bytes belegen die ersten beiden Register im SID.

FL = SI (Frequenz, Lo–Byte)
FH = SI + 1 (Frequenz, Hi–Byte)

Für die Lautstärke sind im SID 16 Stufen vorgesehen, von 0 (ausgeschaltet) bis 15 (volle Lautstärke).
Der entsprechende Parameter wird im Register 24 abgespeichert.

L = SI + 24 (Lautstärke)

Nun kommt die Klangfarbe:

Sie wird im wesentlichen durch die Art der Wellen bestimmt, die das betreffende Musikinstrument erzeugt.

Im C64–Modus sind vier Grundformen vorhanden:

- Dreieck
- Sägezahn
- Rechteck
- Rauschen

In den folgenden Programmierbeispielen werden Sie einige Beispiele lernen, wie man diese Grundformen verändert und durch Filter beeinflussen kann.

Hier reichen uns zunächst die Grundformen: Jede von ihnen wird durch ein Bit im Register 4 kontrolliert:

W = SI + 4 (Wellenform)

In diesem Register schreiben Sie zur Auswahl der o. g. Grundformen einen der Parameter 17,33,65 und 129. Wählen Sie die 65 (die Rechteckwelle),

zusätzlich müssen Sie noch einen Parameter zwischen 0 und 4095 für das sog. Tastverhältnis (das Verhältnis zwischen Ein und Aus Ihres Rechtecks) festlegen.

Beide Bytes dieses Parameters kommen in die Register 2 und 3:

TL = SI + 2 (Tastverhältnis Lo-Byte)
TH = SI + 3 (Tastverhältnis Hi-Byte)

Der Anschlag und der Verlauf des Tones werden in den Registern 5 und 6 eingestellt. Dadurch wird die im Register 24 eingestellte Lautstärke zunächst ansteigen, dann wieder etwas abschwellen.
Die nun erreichte Lautstärke bleibt erhalten, solange Sie den Ton eingeschaltet lassen; dann klingt der Ton aus:

A = SI + 5 (Anschlag)
H = SI + 6 (Halten)

Jedes dieser Register ist in zwei Teile aufgespalten:

Der Parameter in den vier höherwertigen Bits von A regelt das Anklingen des Tons, in den vier niederwertigen Bits das Abschwellen. Kleine Werte bedeuten, hart; große Werte langsam, weich.

Dies gilt ebenfalls für die vier niederwertigen Bits von H, die das Ausklingen des Tones nach seinem Abschalten kontrollieren. Die höheren vier Bits von H bestimmen die Lautstärke, mit der der Ton gehalten wird.

Der höchste Wert ergibt die in Register 24 vorab eingestellte Lautstärke, kleinere Werte schwächen diese Lautstärke mehr oder weniger ab.

Beispielprogramm

Als erstes müssen Sie sich entscheiden, welche Stimmen (Tongeneratoren) Sie benutzen wollen.

Für jede dieser Stimmen müssen Sie dann die vier oben erwähnten Einstellungen (Lautstärke, Wellenform usw.) festlegen. Sie können bis zu drei

Stimmen gleichzeitig verwenden, unser Beispiel benutzt jedoch nur die Stimme Nr. 1.

```
10SI=54272:FL=SI:FH=SI+1:W=SI+4:A=SI+5:H=SI+6:L=SI+24
20POKEL,15
30POKEA,16+9
40POKEH,4*16+4
50POKEFH,29:POKEFL,69
60POKEW,17
70FORT=1TO500:NEXT
80POKEW,0:POKEA,0:POKEH,0
```

In Zeile 10 werden die einzelnen Registeradressen definiert.
In Zeile 20 wird die volle Lautstärke eingestellt.
In Zeile 30 wird der Anschlag eingestellt.
In Zeile 40 wird das Halten und Ausklingen kontrolliert.
In Zeile 50 werden das Hi– und Lo–Byte der Frequenz, hier für den Kammerton A (andere Werte siehe Anhang F) eingestellt.
In Zeile 60 wird die Wellenform reguliert. Dieser POKE–Befehl muß immer als letztes eingestellt werden, da das niedrigste Bit in diesem Register den Tongenerator ein– bzw. ausschaltet.
In Zeile 70 wird die Schleife zur Einstellung der Tondauer eingestellt.

In Zeile 80 werden Wellenform– und Hüllkurveneinstellungen ausgeschaltet.

Nach dem Eingeben von RUN können Sie die Note hören, die mit diesem Programm erzeugt wird.

Melodien im C64–Modus

Sie brauchen kein Musiker zu sein, um im C64–Modus Melodien zu erzeugen. Hier ist ein Programmbeispiel, an dem gezeigt wird, wie man so etwas macht. Wir benutzen wieder nur eine Stimme von den dreien, die uns zur Verfügung stehen.

Löschen Sie mit NEW das vorhergehende Programm und geben Sie folgendes ein:

```
10 REM TONLEITER
20 SI=54272:FL=SI:FH=SI+1:W=SI+4:A=SI+5:H=SI+6:L=SI+24
30 POKE L,15
40 POKE A,9
50 READ X:READ Y
60 IF Y=-1THEN POKE W,0:END
70 POKE FH,X:POKE FL,Y
80 POKE W,17
90 FOR T=1TO100:NEXT
100 POKE W,0
110 FOR T=1TO50:NEXT
120 GOTO40
130 DATA 17,103,19,137,21,237,23,59,26,20,29,69,
        32,219,34,207
140 DATA -1,-1
```

Wenn Sie Töne erzeugen wollen, die einem Cembalo ähneln, müssen Sie die Zeile 80 in folgender Weise abändern:

```
80 POKE W,33
```

Durch diesen POKE–Befehl wählen Sie einen Sägezahn als Wellenform. Dadurch erhalten Sie oberwellenreichere Klänge, als es bei der bisher benutzten Dreieckwellenform der Fall war.

Aber die Wahl der Wellenform ist nur eine der Möglichkeiten, den Klangcharakter zu bestimmen. Durch eine spezielle Wahl des Anschlagwertes können Sie aus dem Cembalo ein Banjo machen. Dies geschieht mit dem nachstehenden POKE–Befehl in Zeile 40.

```
40 POKE A,3
```

Sie können also wie mit einem echten Synthesizer den Klang verschiedener Instrumente nachahmen. Wie das gemacht wird, d.h. in welcher Weise Sie zu diesem Zweck die betreffenden Registerinhalte ändern, werden wir jetzt besprechen.

Wichtige Klangeinstellungen

1. Lautstärke:

Die Wahl der Lautstärke gilt für alle drei Tongeneratoren. Das in Frage kommende Register hat die Adresse 54296. Sie erhalten die max. Lautstärke, wenn Sie in dieses Register einen Wert von 15 poken:

`POKE L,15` oder `POKE 54296,15`

Um die Tongeneratoren auszuschalten, schreiben Sie eine 0 in das Register:

`POKE L,0` oder `POKE 54296,0`

Im allgemeinen stellen Sie die Lautstärke zu Beginn eines Musikprogramms fest ein. Sie können jedoch durch programmierte Änderung der Lautstärke interessante Effekte erzielen.

2. Wellenform:

Wie Sie im oberen Beispiel gesehen haben, bestimmt die Wellenform sehr stark den Klangcharakter eines Tones. Sie können für jede Stimme im C64–Modus die Wellenform getrennt einstellen. Dabei haben Sie die Wahl zwischen:

- Dreieck
- Sägezahn
- Rechteck
- Rauschen

Eine Zusammenstellung der entsprechenden Adressen und ihrer Inhalte, die den verschiedenen Stimmen und Wellenformen entsprechen, gibt die nachstehende Tabelle. Wenn Sie z.B. für die erste Stimme die Wellenform Dreieck wählen wollen, müssen Sie folgenden Befehl anwenden:

`POKE W,17` oder `POKE 54276,17`

Die erste Zahl (Adresse) steht also für das Register. Die zweite Zahl (Inhalt der Adresse oder des Registers) steht für die jeweilige Wellenform.

Die Einstellung der Wellenform

	Register			Inhalt			
STIMME	1	2	3	RAUSCHEN	RECHTECK	SÄGEZAHN	DREIECK
	4	11	18	129	65	33	17

Wir haben diese Tabelle in Zeile 30 unseres Tonleiterprogramms angewendet. Mit `POKE SI+4,17` haben wir das Dreieck als Wellenform gewählt, die wir dann zur Änderung des Klangcharakters durch einen Sägezahn ersetzt haben, indem wir die 17 in eine 33 umgeändert haben.

Als nächstes wollen wir sehen, wie wir die Hüllkurve verändern können, die den Lautstärkeverlauf innerhalb eines Tones bestimmt.
Beachten Sie, daß Sie nur dann einen Ton erhalten, wenn Sie, wie oben beschrieben, auch Lautstärke und Wellenform festlegen.

3. Hüllkurven–Einstellung:

Die Werte für Anschlag und Abschwellen, die wie die Wellenform für jede Stimme getrennt gewählt werden können, werden zusammen durch einen Zahlenwert dargestellt.

Der Anschlagparameter gibt die Zeit an, die der Ton bis zur maximalen (vorher eingestellten) Lautstärke ansteigt.

Der Abschwellparameter ist ein Maß dafür, wie schnell die Lautstärke auf den Haltepegel abfällt. Wurde der Haltepegel 0 gewählt, so ergibt der Abschwellparameter die Abklingzeit bis zur Lautstärke 0 und bestimmt dadurch die Tondauer.

Die den einzelnen Stimmen zugeordneten Adressen und die den verschiedenen Anschlageinstellungen entsprechenden Werte können der folgenden Tabelle entnommen werden:

Anschlag–Einstellungen

	Register			Inhalt	
STIMME	1	2	3	ANSCHLAG	ABSCHWELLEN
	5	12	19	15*16 (weich) ... 0*16 (hart)	15 (weich) ... 0 (hart)

Wenn Sie lediglich eine Anschlagzeit wählen, z.B. durch

```
POKE 54277,64
```

so wird die Abschwellzeit automatisch auf 0 gesetzt.

```
POKE 54277,66
```

stellt den Anschlag auf einen mittleren Wert (64 = 4*16) und das Abschwellen auf einen kleinen Wert (2). Der Wert 66 ergibt sich dann als Summe von 64 + 2. Am besten erkennen Sie den Zusammenhang, wenn Sie statt

```
POKE 54277,66
```

`POKE 54277,4*16+2` schreiben.

Sie sind jetzt an einem Punkt angelangt, wo es am sinnvollsten ist, das Besprochene in einem Programm zusammenzufassen:

```
10 REM EXPERIMENTIERPROGRAMM
20 SI=54272: FL=SI:FH=SI+1:TL=SI+2:TH=SI+3:W=SI+4:
   A=SI+5:H=SI+6:L=SI+24
30 PRINT"DRUECKE EINE TASTE"
40 GET Z$:IF Z$=""THEN 40
50 POKE L+15
60 POKE A,1*16+5
70 POKE H,0*16+0
80 POKE TH,8:POKE TL,0
90 POKE FH,14:POKE FL,162
100 POKE W,65
110 FOR T=1 TO 200:NEXT
120 POKE W=0
130 GOTO 40
```

Zeile 50 – stellt die Lautstärke ein.
Zeile 60 – bestimmt den Anschlag und das Abschwellen.
Zeile 70 – hier wird das Halten und Ausklingen eingestellt.
Zeile 80 – stellt das Tastverhältnis ein.
Zeile 90 – bestimmt die Frequenz.
Zeile 100 – der Generator und die Wellenform werden eingeschaltet.
Zeile 110 – Tondauer.
Zeile 120 – der Generator wird ausgeschaltet.
Zeile 130 – springt nach Zeile 40 und wartet auf einen Tastendruck.

Wir benutzen die Stimme 1 zur Erzeugung eines Tons mit kurzem Anschlag und kurzem Abschwellen nach Erreichen der Maximallautstärke (Zeile 60).

Was dabei herauskommt, klingt etwa wie ein Ball, der in einer Blechtonne hin– und herspringt. Um einen anderen Klang zu erzeugen, ändern wir diese Zeile.

Dazu stoppen wir das Programm und ändern die Zeile 60 wie folgt:

```
60 POKE A,11*16+14
```

Der Ton, den wir mit dieser Einstellung erhalten, hat etwa den Klang einer Oboe oder eines sonstigen Holzblasinstruments.

Experimentieren Sie nun und ändern Wellenform und Hüllkurve, um ein Gefühl dafür zu bekommen, wie die verschiedenen Werte dieser Parameter den Toncharakter verändern.

Mit der Halteeinstellung können Sie festlegen, welche Lautstärke der Ton nach dem Anschlagen beibehält. Die Dauer des Tones wird dabei mit einer FOR. . .NEXT–Schleife geregelt.

Ähnlich wie beim vorigen Register werden das Halten und Ausklingen des Tons durch einen Zahlenwert festgelegt, der sich durch die Addition aus den Werten ermitteln läßt, die in der folgenden Tabelle aufgeführt sind.

Halten und Ausklingen

	Register			Inhalt	
STIMME	1	2	3	HALTEN	AUSKLINGEN
	6	13	20	15*16 (laut) ... 0*16 (stumm)	15 (langsam) ... 0 (schnell)

Ersetzen Sie die Nullen in Zeile 70 durch irgendwelche Werte bis maximal 15 und hören Sie, was dabei herauskommt.

4. Die Wahl der Stimmen und der Noten:

Wie Sie bereits erfahren haben, müssen Sie zur Erzeugung eines Tons zwei Werte eingeben, die wir Hi–Byte und Lo–Byte der Frequenz genannt haben.

Da den Stimmen unterschiedliche Adressen zugeordnet sind (s. folgende Tabelle), können Sie die drei Stimmen im C64–Modus unabhängig voneinander programmieren und auf diese Weise z.B. dreistimmige Musikstücke erstellen.

Adressen der drei Tongeneratoren und POKE–Werte (Hi–Byte und Lo–Byte) der Töne der 5. Oktave.

	Register			Inhalte für Noten der 5. Oktave												
STIMME	1	2	3	C	C#	D	D#	E	F	F#	G	G#	A	A#	H	C
HI-BYTE	1	8	15	35	37	39	41	44	46	49	52	55	58	62	66	70
LO-BYTE	0	7	14	3	24	77	163	29	188	132	117	148	226	98	24	6

Um den Ton C mit der Stimme 1 zu erzeugen, müssen Sie folgende POKE–Befehle geben:

```
POKE54273,35:POKE54272,3
```

oder

```
POKESI+1,35:POKESI,3
```

Denselben Ton mit der 2. Stimme erhalten Sie durch:

```
POKE54280,35:POKE54279,3
```

oder

```
POKESI+8,35:POKESI+7,3
```

Programmieren eines Liedes im C64–Modus

Mit dem folgenden Programmbeispiel kann man Lieder komponieren und wiedergeben. Der Computer benutzt dazu die 1. Stimme. Beachten Sie bitte, daß in der Programmzeile 110 die Adressen der häufig verwendeten Register numerischen Variablen zugeordnet werden und dadurch im Programm bequemer angewendet werden können.

Wenn z.B. die Wellenform gewählt werden soll, so genügt es, im entsprechenden POKE–Befehl den Buchstaben W statt der Zahl 54276 einzusetzen. Weiterhin sollten Sie sich für die Verwendung in eigenen Programmen merken, wie mit den DATA–Zeilen gearbeitet wird. Im vorliegenden Programm werden in den DATA–Zeilen die drei Zahlen, die zur Beschreibung eines Tons notwendig sind, hintereinander abgespeichert.

Es handelt sich hierbei um Hi–Byte und Lo–Byte der Frequenz und die Tondauer. Die Tondauer wird durch eine Schleife bestimmt, die von 1 bis zu einem bestimmten Wert läuft. Dieser Wert steht in den DATA–Zeilen jeweils an 3. Stelle.

Dabei entspricht 125 einer achtel Note, 250 einer viertel Note, 375 einem punktierten Viertel, 500 einer halben und 1000 einer ganzen Note.

Je nach Tempoangabe oder musikalischem Geschmack, können diese Werte natürlich nach oben oder nach unten abgeändert werden.

Schauen wir uns nun mal die Zeile 110 an: Die 17 und die 103 sind Hi–Byte und Lo–Byte für die Note "C". Die Zahl 250 an der dritten Stelle bewirkt, daß es eine viertel Note wird. Auch die zweite Note ist ein Viertel, diesmal ist es aber ein "E" usw.

Sie können in den Datazeilen aber auch selbstgewählte Werte eintragen, die Sie mit Hilfe der Notentabelle im Anhang F ermitteln können. Dabei können Sie Ihre Melodie so lang wählen, wie Speicherplatz im C64–Modus verfügbar ist. Sie müssen nur dafür sorgen, daß die letzte Programmzeile den Ausdruck DATA–1,–1,0 enthält. Die Zeile 130 sorgt dafür, daß das Programm endet, wenn es bei dieser Zeile angelangt ist.

```
10 REM BEISPIELLIED
20 SI=54272: FL=SI:FH=SI+1:TL=SI+2:TH=SI+3:W=SI+4:
   A=SI+5:H=SI+6:L=SI+24
30 POKE L,15:POKE TH,13:POKE TL,15:POKE A,3*16+15:
   POKE H,9
40 READ X:READ Y:READ D
50 IF X=-1 THEN END
60 POKE FH,X:POKE FL,Y
70 POKE W,65
80 FOR T=1 TO D:NEXT
90 POKE W,0
100 GOTO 40
110 DATA 17,103,250,21,237,250,26,20,400,21,237,100,
    26,20,250,29,69,250
120 DATA 26,20,250,0,0,250,21,237,250,26,20,250,29,69,
    1000,26,20,250,0,0,250
130 DATA -1,-1,0
```

Klangeffekte

Im Unterschied zur Musik sollen Klangeffekte Ereignisse, die auf dem Bildschirm stattfinden, untermalen (Explosionen eines Raumschiffs usw.)

oder sie sollen den Benutzer eines Programms informieren oder warnen (z.B., daß er gerade im Begriff ist, eine Diskette zu löschen). Hier einige Vorschläge, die zum Experimentieren anregen sollen:

- Ändern Sie die Lautstärke während der Ton erklingt, so können Sie damit z. B. einen Echoeffekt erzielen.
- Springen Sie schnell zwischen zwei Tönen hin und her, um ein Tremolo zu erzielen.
- Probieren Sie verschiedene Wellenformen aus.
- Beschäftigen Sie sich eingehend mit der Hüllkurve.
- Durch unterschiedliches Programmieren der drei Stimmen (z.B. den Ton in einer Stimme etwas länger anhalten als in der anderen) kann man überraschende Effekte erzielen.
- Benutzen Sie die Rechteckwelle und ändern Sie die Pulsbreite.
- Experimentieren Sie mit dem Rauschgenerator zur Erzeugung von Explosionsgeräuschen, Gewehrfeuer, Schritten usw.
- Ändern Sie in schneller Folge die Frequenz über mehrere Oktaven.

Beispiele für Klangeffekte

Die hier besprochenen Programmbeispiele können Sie in direkter oder in abgeänderter Form in eigene BASIC–Programme einbauen. Sie sollen Sie zum Experimentieren anregen und Ihnen zeigen, welche Klangmöglichkeiten im C64–Modus stecken.

```
10 REM PUPPE
20 SI=54272: FL=SI:FH=SI+1:TL=SI+2:TH=SI+3:W=SI+4:
   A=SI+5:H=SI+6:L=SI+24
30 POKE L,15:POKE TH,15:POKE TL,15:POKE A,0*16+0:
   POKE H,15*16
40 POKE W,65
50 FOR X=250 TO 0 STEP-2:POKE FH,40:POKE FL,X:NEXT
60 FOR X=150 TO 0 STEP-4:POKE FH,40:POKE FL,X:NEXT
70 POKE W,0

10 REM GEWEHRSCHUSS
20 SI=54272:FL=SI:FH=SI+1:TL=SI+2:TH=SI+3:W=SI+4:
   A=SI+5:H=SI+6:L=SI+24
30 FOR X=15 TO 0 STEP-1
40 POKE L,X:POKE A,15:POKE H,0:POKE FH,40:POKE FL,200:
   POKE W,129
50 NEXT
60 POKE W,0:POKE A,0

10 REM MOTOREN
20 SI=54272
30 FOR K=0 TO 24:READ X:POKE SI+K,X:NEXT
40 DATA 9,2,0,3,0,0,240
50 DATA 12,2,0,4,0,0,192
60 DATA 16,2,0,6,0,0,64
70 DATA 0,30,243,31:REM FILTER
80 POKE SI+4,65:POKE SI+11,65:POKE SI+18,65
```

6. Floppy-Disk-Betrieb mit BASIC



6. Floppy–Disk–Betrieb mit BASIC

Der C128 erlaubt den Anschluß von einem oder mehreren Floppy–Disk–Laufwerken zur Speicherung größerer Datenmengen. Einzelheiten zur Bedienung dieser Geräte sind den Floppy–Disk–Handbüchern zu entnehmen. In diesem Abschnitt wird vielmehr der Floppy–Disk–Betrieb im C64– und C128–Modus unter Kontrolle des BASIC 2.0– bzw. BASIC 7.0–Interpreters beschrieben.

Bei der Beschreibung der folgenden Floppy–Disk–Befehle wird davon ausgegangen, daß nur ein Gerät mit einem Laufwerk und der Geräteadresse 8 angeschlossen ist. Sind Doppellaufwerke oder Geräte mit anderen Geräteadressen angeschlossen, so wird auf die Beschreibung der vollständigen Syntax der Floppy–Disk–Befehle in Kapitel 4.6 verwiesen.

Hinweis Die Bezeichnungen 'File' und 'Datei' bedeuten grundsätzlich dasselbe.

6.1 Formatierung von Disketten

Disketten müssen vor dem ersten Gebrauch formatiert werden. Bei diesem Vorgang schreibt das Betriebssystem der Floppy–Disk ein bestimmtes Sektorformat auf die Diskette, das als Rahmen für die später zu speichernden Daten dient. Zusätzlich wird das Disketteninhaltsverzeichnis, das zu Beginn natürlich leer ist, sowie ein spezielles Verzeichnis für freie und belegte Sektoren (BAM) angelegt. Natürlich können auch bereits formatierte Disketten neu formatiert werden. Es ist jedoch eines zu beachten:

- **Bei der Formatierung einer Diskette werden die ggfs. vorher auf dieser Diskette gespeicherten Daten zerstört.**

Formatierung im C128–Modus

- Die zu formatierende Diskette wird in die Floppy–Disk eingelegt (Etikett nach oben und Schreibschutzkerbe links) und das Laufwerk wird verriegelt.

- Folgende Anweisung wird eingegeben:

HEADER *Name*, I*Kennung*, D*Laufw*

wobei für ***Name*** eine bis zu 16 Zeichen lange Zeichenkette eingekleidet in Anführungszeichen (") und für ***Kennung*** eine 2 Zeichen lange Zeichenkette eingegeben wird (siehe dazu Beschreibung des HEADER–Befehls in Kapitel 4.6). Für ***Laufw*** wird 0 oder 1 angegeben.

- BASIC fordert vor der Ausführung der Formatierung mit der Anfrage

```
ARE YOU SURE?
```

vom Anwender eine Bestätigung. Hier muß **Y** oder **YES** eingegeben werden. Bei jeder anderen Antwort wird die Formatierung nicht ausgeführt.

- Während der Formatierung verschwindet der Cursor.

- Nach erfolgter Formatierung meldet sich BASIC mit READY. Jeglicher während der Formatierung von DOS diagnostizierte Fehler wird durch die Meldung

```
BAD DISK
```

angezeigt. Den tatsächlich vom DOS diagnostizierten Fehler kann man sich durch den Befehl **PRINT DS$** anzeigen lassen. In diesem Fall muß die Formatierung mit einer neuen Diskette wiederholt werden.

Formatierung im C64–Modus

- Die zu formatierende Diskette wird in die Floppy–Disk eingelegt (Etikett nach oben und Schreibschutzkerbe links) und die Laufwerkstür wird verriegelt.

- Folgende Anweisungen werden eingegeben:

```
OPEN 15,8,15,"N0:Name,Kennung":CLOSE15
```

wobei für ***Name*** eine bis zu 16 Zeichen lange Zeichenkette und für ***Kennung*** eine 2 Zeichen lange Zeichenkette eingegeben wird.

- BASIC meldet sich sofort wieder mit READY. Die Formatierung war erfolgreich, wenn die rote oder grüne Aktivitätslampe (LED) der Floppydisk erlischt. Blinkt sie, so hat das DOS einen Fehler diagnostiziert.

- Für die Fehlerabfrage im C64–Modus muß folgendes kleine BASIC–Programm geschrieben werden:

```
10 OPEN 15,8,15:INPUT#15,FC,FM$,FS,FS
20 PRINT FC;FM$FS;FT:CLOSE15
```

6.2 Speicherung von Programmen

Speicherung im C128–Modus

Im C128–Modus lassen sich sowohl BASIC– als auch in anderen Speicherbereichen abgelegte Maschinensprache–Programme speichern.

Bei BASIC–Programmen wird folgender Befehl eingegeben:

DSAVE *Name*

Für ***Name*** wird eine bis zu 16 Zeichen lange Zeichenkette, eingekleidet in Anführungsstrichen ("), angegeben.

Die erfolgreiche Speicherung wird angezeigt, wenn BASIC sich mit READY. meldet und die rote oder grüne Aktivitätslampe am Laufwerk erlischt. Blinkt die Anzeige, so hat DOS einen Fehler diagnostiziert, der mit **PRINT DS$** angezeigt werden kann.

Unmittelbar nach der Speicherung kann das gespeicherte Programm byteweise mit dem im Hauptspeicher befindlichen Programm mit Hilfe des Befehls **DVERIFY *Name*** verglichen werden. Eventuelle Fehler werden angezeigt.

Bei Maschinenspracheprogrammen, die in anderen Speicherbereichen als dem BASIC–Programmspeicher abgelegt sind, wird folgender Befehl eingegeben.

BSAVE *Name* ON B*Bank*, P*Adr1* TO P*Adr2*

Für ***Name*** wird eine bis zu 16 Zeichen lange Zeichenkette in Anführungszeichen ("), für ***Bank*** die Nummer der Speicherbank, aus der das Programm gespeichert werden soll (0, 1 oder 15) und für ***Adr1*** und ***Adr2*** Anfangs– und Endadresse des zu speichernden Programms als ganze Dezimalzahlen eingegeben.

Speicherung im C64–Modus

Im C64–Modus lassen sich nur BASIC–Programme auf Floppy–Disk speichern. Dazu wird folgender Befehl eingegeben:

```
SAVE"0:Name",8
```

Für ***Name*** wird eine bis zu 16 Zeichen lange Zeichenkette angegeben.

Die erfolgreiche Speicherung wird angezeigt, wenn BASIC sich mit READY. meldet und die rote oder grüne Aktivitätslampe am Laufwerk erlischt. Blinkt die Anzeige, so hat das DOS einen Fehler diagnostiziert, der wie in Kapitel 7.1 beschrieben, angezeigt werden kann.

Unmittelbar nach der Speicherung kann das gespeicherte Programm byteweise mit dem im Hauptspeicher befindlichen Programm mit Hilfe des Befehls VERIFY *Name*,8 verglichen werden. Eventuelle Fehler werden angezeigt.

6.3 Verwendung von Joker–Zeichen in Dateinamen

Bei einer Reihe von Floppy–Disk–Befehlen (OPEN, DOPEN, LOAD, DLOAD, SCRATCH, VERIFY, DVERIFY, COPY) kann der dort erforderliche Dateiname auch in abgekürzter Form angegeben werden. Dazu stehen zwei sogenannte Joker–Zeichen, nämlich

* und **?**

zur Verfügung.

* kann nur am Ende eines Teilnamens stehen und bedeutet, daß alle weiteren Zeichen des Namens irrelevant sind, daß also der Befehl auf die erste Datei bzw. alle weiteren im Diskettenverzeichnis wirkt, deren Name mit dem angegebenen Teilnamen beginnt:

`DLOAD"ADR*"`	Lädt das erste Programm aus dem Diskettenverzeichnis, dessen Name mit den Zeichen ADR beginnt.
`DLOAD"*"`	Lädt das erste Programm aus dem Diskettenverzeichnis.

? steht für ein beliebiges Zeichen an dieser Stelle:

`DLOAD "DATE?"` Lädt das erste Programm aus dem Diskettenverzeichnis, dessen Name 5 Zeichen lang ist und das mit DATE beginnt.

`DLOAD "PRO??.*"` Lädt das erste Programm aus dem Diskettenverzeichnis, dessen Name mit PRO beginnt, dann 2 beliebige Zeichen und auf der 6 Position einen Punkt enthält.

6.4 Laden von Programmen

Laden im C128–Modus

Im C128–Modus lassen sich sowohl BASIC– als auch Maschinensprache-Programme (auch in andere Speicherbereiche) laden.

Bei BASIC–Programmen wird folgender Befehl eingegeben:

DLOAD *Name*

Für ***Name*** wird eine bis zu 16 Zeichen lange Zeichenkette, eingekleidet in Anführungsstrichen ("), angegeben.

Der erfolgreiche Ladevorgang wird angezeigt, wenn BASIC sich mit READY. meldet und die rote oder grüne Aktivitätslampe am Laufwerk erlischt. Blinkt die Anzeige, so hat das DOS einen Fehler diagnostiziert, der mit **PRINT DS$** angezeigt werden kann.

Bei Maschinenspracheprogrammen, die in andere Speicherbereiche als den BASIC–Programmspeicher geladen werden sollen, wird folgender Befehl eingegeben:

BLOAD *Name* ON B*Bank*, P*Adr*

Für ***Name*** wird eine bis zu 16 Zeichen lange Zeichenkette in Anführungszeichen ("), für ***Bank*** die Nummer der Speicherbank, in die das Programm geladen werden soll (0 oder 1) und für ***Adr*** die Adresse als ganze Dezimalzahl, ab der geladen werden soll, angegeben. Der weitere Ablauf ist dann wie bei DLOAD beschrieben.

Laden im C64–Modus

Auch im C64–Modus lassen sich sowohl BASIC–Programme als auch Maschinenspracheprogramme laden.

Bei BASIC–Programmen wird folgender Befehl eingegeben:

```
LOAD"Name",8
```

Für ***Name*** wird eine bis zu 16 Zeichen lange Zeichenkette, eingekleidet in Anführungsstrichen ("), angegeben.

Die erfolgreiche Speicherung wird angezeigt, wenn BASIC sich mit READY. meldet und die rote oder grüne Aktivitätslampe am Laufwerk erlischt. Blinkt die Anzeige, so hat das DOS einen Fehler diagnostiziert, der wie in Kapitel 7.1 beschrieben, angezeigt werden kann.

Bei Maschinensprache–Programmen wird folgender Befehl eingegeben:

```
LOAD Name,8,1
```

Hier muß beachtet werden, daß das Programm mit diesem Befehl nur ab der Adresse geladen wird, ab der es auch ursprünglich gespeichert wurde (z.B. mit Hilfe eines speziellen Maschinensprache–Monitors). Der weitere Ladevorgang läuft dann, wie oben bei LOAD beschrieben, ab.

6.5 Diskettenverzeichnisse

Ein Diskettenverzeichnis ist eine Liste, in der alle auf der Diskette gespeicherten Programm– und Datendateien verzeichnet sind. Diese Liste kann sich der Anwender jederzeit auf dem Bildschirm anzeigen lassen oder auch in den Speicher laden. Ebensogut können Auszüge aus dem Verzeichnis gemacht werden.

Format des Diskettenverzeichnisses

Das DOS liefert bei Abruf das Diskettenverzeichnis in einem ganz bestimmten Format. Der erste Eintrag enthält links die Laufwerknummer (0 oder 1), dann den bei der Formatierung angegebenen Diskettennamen in Anführungszeichen, dann die bei der Formatierung angegebene Diskettenkennung (2 Zeichen) und schließlich die Nummer der DOS–Version, unter der die Diskette formatiert wurde.

Die folgenden Einträge sind Dateieinträge. Sie enthalten links die Dateigröße in Sektoren oder Blöcken (1 Sektor enthält maximal 254 Datenbytes), dann den Dateinamen in Anführungszeichen und schließlich den Dateityp. Das DOS unterscheidet 4 Dateitypen, nämlich:

PRG für Programmdateien;
SEQ für sequentielle Datendateien;
REL für relative Datendateien;
USR für anwenderspezifische Datendateien;

Steht unmittelbar vor dem Dateityp ein Stern (*), so ist diese Datei nicht ordnungsgemäß geschlossen und es kann nicht auf die Daten dieser Datei zugegriffen werden.

Achtung
Direktzugriffsdateien, die vom Anwender mit den Blockbefehlen bearbeitet werden, werden vom DOS nicht automatisch im Diskettenverzeichnis eingetragen. Das Floppy–Disk–Handbuch enthält detaillierte Beschreibungen über die Behandlung von Direktzugriffs– und von Relativ–Dateien.

Der letzte Eintrag im Verzeichnis enthält die Anzahl der noch nicht belegten Sektoren oder Blöcke auf der Diskette.

Ein typisches Diskettenverzeichnis sieht so aus:

```
0    "DISKETTE 1      " D1 2A
15   "ADRESSDRUCK"       PRG
122  "ADRESSEN"          SEQ
85   "FIBU"              PRG
290  "STAMMDAT"          REL
152 BLOCKS FREE
```

Diskettenverzeichnisanzeigen im C128–Modus

`DIRECTORY`	zeigt das gesamte Diskettenverzeichnis auf dem Bildschirm an.
`DIRECTORY "AD*"`	zeigt alle Dateieinträge an, deren Namen mit AD beginnen.

`DIRECTORY"F*=P"`	zeigt alle Programmdateieinträge an, deren Namen mit F beginnen.
`DIRECTORY"*=R"`	zeigt alle Relativdateieinträge an.

Enthält das Verzeichnis mehr Einträge, als auf einen Bildschirm passen, so kann die Anzeige mit Hilfe der Commodore–Taste verlangsamt oder mit der NO SCROLL–Taste angehalten werden.

Diskettenverzeichnisanzeigen im C64–Modus

Im C64–Modus muß das Diskettenverzeichnis vor der Anzeige in den Speicher geladen werden. Ein ggfs. vorher dort gespeichertes Programm wird dadurch überschrieben. Nach dem Laden wird das Verzeichnis wie ein Programm mit dem Befehl LIST auf dem Bildschirm angezeigt.

`LOAD"$0",8`	lädt das gesamte Diskettenverzeichnis in den Speicher.
`LOAD"$0:AD*",8`	lädt alle Dateieinträge in den Speicher, deren Namen mit AD beginnen.
`LOAD"$0:F*=P",8`	lädt alle Programmdateieinträge in den Speicher, deren Namen mit F beginnen.
`LOAD"$0:*=R",8`	lädt alle Relativdateieinträge in den Speicher.

Das geladene Verzeichnis wird dann mit dem Befehl LIST auf dem Bildschirm dargestellt. Das Listen kann nicht angehalten, wohl aber durch Drücken der CTRL–Taste verlangsamt werden.

Ausdruck des Diskettenverzeichnisses

In beiden Betriebsmodi kann ein Diskettenverzeichnis mit folgender Befehlsfolge auf einem angeschlossenen Drucker mit der Geräteadresse 4 ausgedruckt werden.

```
LOAD"$0",8
OPEN4,4:CMD4:LIST
PRINT#4:CLOSE4
```

Sollen nur Auszüge des Verzeichnisses ausgedruckt werden, so ist die oben beim C64–Modus beschriebene Form des LOAD–Befehls zu verwenden.

6.6 Öffnen und Schließen von Dateien

Um Daten in BASIC mit der PRINT#–Anweisung (s. dort in Kapitel 5.2) in eine Datei auf Diskette zu schreiben oder mit der GET#– oder INPUT#–Anweisung (s. dort in Kapitel 5.2) aus einer Datei auf Diskette zu lesen, muß die Datei zunächst geöffnet und nach dem Bearbeiten wieder geschlossen werden. Dafür werden im folgenden Beispiele, getrennt für die beiden Betriebsmodi, gegeben.

Datei öffnen im C128–Modus

`DOPEN #3,"ADRESSEN"`

Öffnet die sequentielle Eingabe–Datei ADRESSEN (entweder SEQ– oder USR–Typ) unter der logischen Filenummer 3.

`DOPEN #2,"STAMMDAT",W`

Öffnet die sequentielle Ausgabe-Datei STAMMDAT unter der logischen Filenummer 2.

`DOPEN #7,"BANK",L127`

Öffnet die Relativ–Datei BANK mit einer Satzlänge von 127 Bytes unter der logischen Filenummer 7.

`OPEN 4,8,4,"0:PROG,P,R"`

Öffnet die Programm–Datei PROG als Eingabe–Datei unter der logischen Filenummer 4.

`OPEN5,8,4,"0:GRAFIK,U,W"`

Öffnet die Datei GRAFIK als anwenderspezifische, sequentielle Ausgabedatei unter der logischen Filenummer 5.

`APPEND #3,"NAMEN"`	Öffnet die existierende, sequentielle Datei NAME (SEQ– oder USR–Typ) unter der logischen Filenummer 3 und positioniert den Schreibzeiger des DOS auf das Dateiende, damit weitere Daten angefügt werden können.
`OPEN 3,8,4,"0:ADRESSEN,S,R"`	Öffnet die sequentielle Eingabe–Datei ADRESSEN unter der logischen Filenummer 3.
`OPEN 2,8,5,"0:STAMMDAT,S,W"`	Öffnet die sequentielle Ausgabe-Datei STAMMDAT unter der logischen Filenummer 2.
`OPEN 7,8,2,"0:BANK,L,"+CHR$(127)`	Öffnet die Relativ–Datei BANK mit einer Satzlänge von 127 Bytes unter der logischen Filenummer 7.
`OPEN 4,8,4,"0:PROG,P,R"`	Öffnet die Programm–Datei PROG als Eingabe–Datei unter der logischen Filenummer 4.
`OPEN 5,8,4,"0:GRAFIK,U,W"`	Öffnet die Datei GRAFIK als anwenderspezifische, sequentielle Ausgabedatei unter der logischen Filenummer 5.
`OPEN 3,8,3,"0:NAMEN,A"`	Öffnet die existierende, sequentielle Datei NAME (SEQ– oder USR–Typ) unter der logischen Filenummer 3 und positioniert den Schreibzeiger des DOS auf das Dateiende, damit weitere Daten angefügt werden können.

Wird eine geöffnete Datei in einem Programm nicht mehr benötigt, so muß sie vor dem Programmende oder einem Diskettenwechsel wieder geschlossen werden. Bei sequentiellen Ausgabedateien (SEQ– oder USR–Typ) ist dies unerläßlich, da sonst auf die gespeicherten Daten nicht mehr zugegriffen werden kann. Solche Dateien werden im Verzeichnis durch einen Stern (*) vor dem Dateityp gekennzeichnet.

Datei schließen im C128–Modus

`DCLOSE #5`	schließt die unter der logischen Filenummer 5 geöffnete Datei beliebigen Typs.
`DCLOSE`	schließt alle geöffneten Diskettendateien.
`CLOSE 5`	schließt die unter der logischen Filenummer 5 geöffnete Datei beliebigen Typs.

Einen Befehl, mit dem man im C64–Modus alle geöffneten Dateien schließen kann, gibt es im BASIC 2.0 nicht.

6.7 Schreib–/Lesezeiger in Relativdateien positionieren

Ehe auf einen logischen Datensatz in einer Relativdatei zugegriffen werden kann (mit PRINT# bzw. GET# oder INPUT#), muß der Schreib–/Lesezeiger des DOS für diese Datei auf den gewünschten Satz positioniert werden

Relativdateizugriff im C128–Modus

`RECORD #2,200`	Der Schreib–/Lesezeiger der Datei, die unter der logischen Filenummer 2 eröffnet wurde, wird auf den 200. Datensatz positioniert.

```
RECORD #2,200,50
```

Wie oben. Zusätzlich wird auf das 50. Byte des 200. Satzes positioniert.

Während der Positionierung brennt die rote oder grüne Aktivitätslampe der Floppy–Disk. Blinkt sie, so hat das DOS einen Fehler diagnostiziert, der mit der Anweisung PRINT DS$ angezeigt werden kann.

Relativdateizugriff im C64–Modus

Im C64–Modus erfolgt der Relativdateizugriff über einen speziellen Befehl, der über den Befehlskanal 15 der Floppy–Disk gesendet wird.

Dieser Befehl hat die folgende allgemeine Form:

```
"P"+CHR$(SA)+CHR$(SN)+CHR$(SH)+CHR$(BN)
```

SA ist die bei der Dateieröffnung angegebene Sekundäradresse (Kanalnummer), die noch um 96 erhöht werden muß.

SN ist der niederwertige Teil der Satznummer *S*, auf die positioniert werden soll. *SN*=*S*–INT(*S*/256)*256.

SH ist der höherwertige Teil der Satznummer *S*, auf die positioniert werden soll. *SH*=INT(*S*/256).

BN ist die Bytenummer innerhalb des gewünschten Satzes, auf die positioniert werden soll.

Bei den folgenden Beispielen wird angenommen, daß die Datei mit der folgenden Anweisung eröffnet wurde:

```
OPEN 7,8,2,"0:BANK,L,"+CHR$(127):OPEN 15,8,15
```

```
PRINT #15,"P"+CHR$(2+96)+
          CHR$(200)+CHR$(0)+
          CHR$(1)
```

Der Schreib–/Lesezeiger der Datei, die unter der logischen Filenummer 2 eröffnet wurde, wird auf die erste Position des Satzes Nummer 200 gestellt.

```
PRINT #15,"P"+CHR$(2+96)+
        CHR$(194)+CHR$(1)+
        CHR$(50)
```

Der Schreib–/Lesezeiger der Datei, die unter der logischen Filenummer 2 eröffnet wurde, wird auf die 50. Position des Satzes Nummer 450 gestellt.

Während der Positionierung brennt die rote oder grüne Aktivitätslampe der Floppy–Disk. Blinkt sie, so hat DOS einen Fehler diagnostiziert, der mit der in Kapitel 6.1 beschriebenen Anweisungsfolge angezeigt werden kann.

6.8 Löschen von Diskettendateien

Jede im Diskettenverzeichnis aufgeführte Datei kann vom Anwender gelöscht werden.

Datei löschen im C128–Modus

Es wird folgender Befehl eingegeben:

SCRATCH *Name*

Für ***Name*** wird eine bis zu 16 Zeichen lange Zeichenkette, eingekleidet in Anführungsstrichen (") angegeben. Joker–Zeichen (s. Kapitel 7.3) sind dabei zum Löschen mehrerer Dateien mit gleichen Namensteilen erlaubt.

Ehe der Befehl ausgeführt wird, fordert der Interpreter vom Anwender mit der Anfrage

```
ARE YOU SURE?
```

eine Bestätigung. Hier muß **Y** oder **YES** eingegeben werden. Bei jeder anderen Antwort wird der Löschvorgang nicht ausgeführt.

Während der Löschung brennt die rote oder grüne Aktivitätslampe der Floppy–Disk. Das DOS meldet die erfolgreiche Löschung mit der Meldung

```
01, FILES SCRATCHED,nn,00
```

wobei für *nn* die Anzahl der gelöschten Dateien gesetzt wird. Blinkt die Aktivitätslampe, so hat DOS beim Löschen einen Fehler diagnostiziert, der mit der Anweisung **PRINT DS$** angezeigt werden kann.

Datei löschen im C64–Modus

Es wird folgende Anweisungsfolge eingegeben:

OPEN 15,8,15,"S0:*Namen*":CLOSE 15

Für ***Namen*** wird eine bis zu 37 Byte lange Zeichenkette eingegeben, die den Namen oder die Namen, durch Komma getrennt, der zu löschenden Datei(en) enthält. Joker–Zeichen (s. Kapitel 7.3) sind dabei zum Löschen mehrerer Dateien mit gleichen Namensteilen erlaubt.

Die Anzahl der tatsächlich gelöschten Dateien kann mit der in Kapitel 7.1 beschriebenen Fehlerabfrage angezeigt werden. Das gleiche gilt für Fehler, die das DOS während des Löschvorganges diagnostiziert hat, und die durch eine blinkende Aktivitätslampe am Laufwerk angezeigt werden.

6.9 Weitere Verwaltungsfunktionen

6.9.1 Diskettenbelegung überprüfen und bereinigen

Nach einer längeren Betriebszeit kann es vorkommen, daß eine häufig benutzte Diskette, bei der wiederholt Dateien gelöscht und neu geschrieben wurden, eine Anzahl von Sektoren enthält, die nicht benutzt werden, obwohl sie eigentlich als frei gekennzeichnet sein müßten. Um diese Sektoren wieder verfügbar zu machen und alle Dateien, die nicht ordnungsgemäß geschlossen wurden (Stern vor dem Dateityp im Diskettenverzeichnis), zu löschen, kann der COLLECT–Befehl verwendet werden.

Wird der COLLECT–Befehl im C128–Modus bei noch offenen Dateien verwendet, so werden diese Dateien ordnungsgemäß geschlossen.

Diskette bereinigen im C128–Modus

Es wird folgender Befehl eingegeben:

COLLECT

Da nach Beendigung der Operation das Verzeichnis über verfügbare Sektoren (BAM) neu geschrieben wird, darf die betreffende Diskette nicht mit einem Schreibschutz versehen sein.

Diskette bereinigen im C64–Modus

Es wird folgende Anweisungsfolge eingegeben:

OPEN 15,8,15,"V0":CLOSE 15

6.9.2 Datei kopieren

Eine Datei kann innerhalb eines Laufwerkes in eine andere Datei mit anderem Namen kopiert werden. Bei Verwendung von Doppellaufwerken kann die Datei unter dem selben Namen auf die Diskette im anderen Laufwerk kopiert werden (s. Floppy–Disk–Handbuch).

Datei kopieren im C128–Modus

Es wird folgender Befehl eingegeben:

COPY *Name1* TO *Name2*

Für ***Name1*** (Quelldatei) und ***Name2*** (Zieldatei) werden bis zu 16 Zeichen lange Zeichenketten, eingekleidet in Anführungszeichen ("), angegeben.

Während des Kopiervorganges brennt die rote oder grüne Aktivitätslampe der Floppy–Disk. Blinkt sie, so hat das DOS beim Kopieren einen Fehler diagnostiziert, der mit der Anweisung **PRINT DS$** angezeigt werden kann.

Datei kopieren im C64–Modus

Es wird folgende Anweisungsfolge eingegeben:

OPEN 15,8,15,"C0:*Name2*= 0:*Name1*"

Für ***Name2*** (Zieldatei) und ***Name1*** (Quelldatei) werden Zeichenketten von bis zu 16 Zeichen Länge eingegeben. Der Zieldateiname wird als erster angegeben. Eventuell von DOS diagnostizierte Fehler beim Kopiervorgang können mit der in Kapitel 7.1 beschriebenen Fehlerabfrage angezeigt werden.

6.9.3 Verkettung sequentieller Dateien

Zwei sequentielle Dateien können durch einen besonderen Kopierbefehl aneinandergehängt werden.

Dateiverkettung im C128–Modus

Es wird folgender Befehl eingegeben:

CONCAT *Name1* TO *Name2*

Für ***Name1*** (Quelldatei) und ***Name2*** (Zieldatei) werden bis zu 16 Zeichen lange Zeichenketten, eingekleidet in Anführungszeichen ("), angegeben. Durch diesen Befehl wird an die existierende Datei ***Name2*** die Datei ***Name1*** angehängt.

Während des Kopiervorganges brennt die rote oder grüne Aktivitätslampe der Floppy–Disk. Blinkt sie, so hat DOS beim Kopieren einen Fehler diagnostiziert, der mit der Anweisung **PRINT DS$** angezeigt werden kann.

Dateiverkettung im C64–Modus

Es wird folgende Anweisungsfolge eingegeben:

OPEN 15,8,15,"C0:*Name3=0:Name1*,0:*Name2*"

Für ***Name3*** (Zieldatei), ***Name1*** (Quelldatei 1) und ***Name2*** (Quelldatei 2) werden bis zu 16 Zeichen lange Zeichenketten eingegeben. Beim Kopiervorgang wird zunächst die Quelldatei 1 in die Zieldatei kopiert und anschließend die Quelldatei 2 angefügt. Für ***Name3*** und ***Name1*** kann derselbe Dateiname verwendet werden. In diesem Fall würde die Datei ***Name2*** an das Ende von der Datei ***Name1*** angehängt. Eventuell von DOS diagnostizierte Fehler beim Kopiervorgang können mit der in Kapitel 7.1 beschriebenen Fehlerabfrage angezeigt werden.

6.9.4 Datei umbenennen

Jeder Datei im Diskettenverzeichnis kann ein neuer, noch nicht benutzter Name zugewiesen werden.

Datei umbenennen im C128–Modus

Es wird folgender Befehl eingegeben:

RENAME *Namealt* TO *Nameneu*

Für ***Nameneu*** und ***Namealt*** werden bis zu 16 Zeichen lange Zeichenketten, eingekleidet in Anführungszeichen ("), angegeben.

Während der Umbenennung brennt die rote oder grüne Aktivitätslampe der Floppy–Disk. Blinkt sie, so hat DOS beim Umbenennen einen Fehler diagnostiziert, der mit der Anweisung **PRINT DS$** angezeigt werden kann.

Datei umbenennen im C64–Modus

Es wird folgende Anweisungsfolge eingegeben:

OPEN 15,8,15,"R0:*Nameneu*=0:*Namealt*"

Für ***Nameneu*** und ***Namealt*** werden bis zu 16 Zeichen lange Zeichenketten angegeben. Der neue Name wird zuerst angegeben. Eventuell vom DOS diagnostizierte Fehler beim Umbennen können mit der in Kapitel 7.1 beschriebenen Fehlerabfrage angezeigt werden.

6.9.5 Diskette duplizieren

Disketten können mit dem Duplizierbefehl nur kopiert werden, wenn ein Floppy–Disk–Doppellaufwerk zur Verfügung steht.

In das eine Laufwerk wird die zu duplizierende Diskette und in das andere Laufwerk eine Leerdiskette eingelegt.

Um Datenverluste durch Verwechseln von Quell– und Ziellaufwerk beim Duplizierbefehl zu vermeiden, sollte die Quelldiskette mit einer Schreibschutzlasche versehen werden.

Diskettenduplizierung im C128–Modus

Es wird folgender Befehl eingegeben:

BACKUP D*Quellaufwerk* TO D*Ziellaufwerk*

Für ***Quellaufwerk*** und ***Ziellaufwerk*** werden 0 und 1 oder 1 und 0 eingegeben, je nachdem, in welchem Laufwerk welche Diskette eingelegt wurde.

Da die Zieldiskette vor dem Kopiervorgang neu formatiert wird, wird der Anwender mit der Anfrage

```
ARE YOU SURE?
```

zur Bestätigung aufgefordert. Anwortet er mit **Y** oder **YES**, so wird der Dupliziervorgang gestartet, bei jeder anderen Antwort abgebrochen.

Jeder beim Dupliziervorgang vom DOS diagnostizierte Fehler wird mit der Bildschirmmeldung

```
BAD DISK
```

angezeigt. Der tatsächliche Fehler kann dann mit **PRINT DS$** angezeigt werden.

Diskette duplizieren im C64–Modus

Es wird folgende Anweisungsfolge eingegeben:

OPEN 15,8,15,"D*Ziellaufw*= D*Quellaufw*":CLOSE 15

Für ***Ziellaufw*** und ***Quellaufw*** werden 0 und 1 oder 1 und 0 eingegeben, je nachdem, in welchem Laufwerk welche Diskette eingelegt wurde. Das Ziellaufwerk wird zuerst angegeben.

Während des Dupliziervorganges brennt die rote oder grüne Aktivitätslampe der Floppy-Disk. Blinkt sie, so hat DOS einen Fehler diagnostiziert, der mit der in Kapitel 7.1 beschriebenen Fehlerabfrage angezeigt werden kann.

8. Fehlermeldungen



8. Fehlermeldungen

8.1 BASIC–Fehlermeldungen

In diesem Abschnitt werden die Fehlermeldungen der BASIC–Interpreter 2.0 und 7.0 und ihre Fehlercodes (nur BASIC 7.0) aufgeführt. Zur schnelleren Orientierung werden zunächst die Fehlermeldungen in alphabetischer Reihenfolge angegeben, anschließend nach Fehlercodes geordnet.

Jeder Fehlermeldung wird bei der Anzeige stets das Wort ERROR (Fehler) hinzugefügt.

Im C64–Modus gibt es keine Fehlercodes und damit auch keine Möglichkeit der programmierten Fehlerverarbeitung. D.h., in diesem Modus führt ein Fehler auf jeden Fall zum Abbruch des laufenden Programms.

Im C128–Modus kann mit Hilfe der Systemvariablen EL und ER, der Funktion ERR$ sowie den Anweisungen TRAP und RESUME (s. dort in Kapitel 4.6) eine programmierte Fehlerverarbeitung realisiert werden.

Code	Meldung und Bedeutung
36	**BAD DISK** (Schlechte Diskette) Es wurde entweder versucht, eine noch nicht formatierte Diskette mit dem verkürzten HEADER–Befehl (s. dort in Kapitel 4.6) zu löschen oder eine schadhafte Diskette zu formatieren.
18	**BAD SUBSCRIPT** (Falscher Feldindex) Es wird versucht, auf ein Feldlelement zuzugreifen, das ausserhalb des in der entsprechenden DIM–Anweisung festgelegten Bereiches liegt.
30	**BREAK** (Unterbrechung) Die STOP–Taste wurde gedrückt, um das laufende Programm zu unterbrechen.
26	**CAN'T CONTINUE** (keine Fortsetzung möglich) Der CONT–Befehl ist nur wirksam, wenn das Programm bereits einmal gelaufen ist und dann unterbrochen wurde. Während der Unterbrechung darf das Programm jedoch nicht verändert werden (z.B. durch Hinzufügen weiterer Zeilen).

BASIC–Fehlermeldungen (Fortsetzung)

Code	Meldung und Bedeutung
31	**CAN'T RESUME** (RESUME ohne TRAP) Der Interpreter findet eine RESUME–Anweisung, ohne daß eine TRAP–Anweisung zur Fehlerverzweigung vorhanden ist.
5	**DEVICE NOT PRESENT** (Gerät nicht angeschlossen) Ein Ein–/Ausgabe–Gerät wird angesprochen, das entweder ausgeschaltet oder nicht angeschlossen ist, oder die Pufferzuweisung für Kassetten–Ein–/Ausgabe ist aufgehoben.
34	**DIRECT MODE ONLY** (nur Direktmodus erlaubt) Der Interpreter erkennt während der Abarbeitung eines Programms einen Befehl oder eine Anweisung, der/die nur im Direktmodus erlaubt ist.
20	**DIVISION BY ZERO** (Division durch Null) Eine Division durch den Wert 0 ist nicht zulässig, da das Ergebnis unendlich wird.
24	**FILE DATA** (ungültige Dateidaten) Es wird versucht, mit einer INPUT–Anweisung Zeichenkettendaten von einer Eingabedatei einer numerischen Variablen zuzuweisen.
4	**FILE NOT FOUND** (Datei nicht gefunden) Es wird entweder die spezifizierte Datei auf Diskette nicht gefunden oder die Bandendemarke auf Kassette wird gelesen.
3	**FILE NOT OPEN** (Datei nicht geöffnet) Es wird eine Ein– oder Ausgabeoperation für eine Datei versucht, die vorher nicht geöffnet wurde.
2	**FILE OPEN** (Datei bereits geöffnet) Es wird versucht, mit einer logischen Dateinummer eine Datei zu öffnen, unter der bereits vorher eine andere Datei geöffnet wurde.
25	**FORMULA TOO COMPLEX** (zu komplexer Ausdruck) Es wird ein zu komplexer numerischer oder Zeichenkettenausdruck gefunden. Ausdruck teilen oder weniger Klammern verwenden.
9	**ILLEGAL DEVICE NUMBER** (unerlaubte Geräteadresse) Es wird entweder versucht eine Ein–/Ausgabe–Operation mit einem unerlaubten Gerät oder Einheit auszuführen (z.B. SAVE zum Bildschirm) oder es wird eine Gräteadresse oberhalb von 15 verwendet.

BASIC–Fehlermeldungen (Fortsetzung)

Code	Meldung und Bedeutung
21	**ILLEGAL DIRECT** (unerlaubter Eingabemodus) Es wird versucht, mit INPUT oder GET im Direktmodus Daten einzulesen.
14	**ILLEGAL QUANTITY** (unerlaubter Wert) Ein numerisches Argument einer Funktion oder ein numerischer Parameter für einen Befehl oder eine Anweisung liegt außerhalb des zulässigen Bereiches.
29	**LOAD** (Ladefehler) Beim Laden eines Programms von Kassette oder Diskette ist ein Lesefehler aufgetreten. Ein erneuter Versuch sollte zunächst gemacht werden.
32	**LOOP NOT FOUND** (DO ohne LOOP) Der Interpreter findet zu einer vorhandenen DO–Anweisung nicht die zugehörige LOOP–Anweisung.
33	**LOOP WITHOUT DO** (LOOP ohne DO) Der Interpreter findet eine LOOP–Anweisung ohne vorausgegangene DO–Anweisung.
8	**MISSING FILE NAME** (Dateiname fehlt) In einem Befehl oder einer Anweisung fehlt die vorgeschriebene Dateiangabe.
10	**NEXT WITHOUT FOR** (NEXT ohne FOR) Der Interpreter findet eine NEXT–Anweisung, zu der keine vorausgegangene FOR–Anweisung existiert.
35	**NO GRAPHICS AREA** (kein Grafikbereich reserviert) Es wird versucht, eine grafische Anweisung (BOX, CIRCLE, DRAW usw.) auszuführen, ohne daß vorher mit der GRAPHIC–Anweisung Speicherbereich reserviert wurde.
6	**NOT INPUT FILE** (keine Eingabedatei) Es wird versucht, mit INPUT oder GET aus einer Datei Daten zu lesen, die als Ausgabedatei geöffnet wurde.
7	**NOT OUTPUT FILE** (keine Ausgabedatei) Es wird versucht, mit PRINT oder CMD Daten in eine Datei auszugeben, die als Eingabedatei geöffnet wurde.

BASIC–Fehlermeldungen (Fortsetzung)

Code	Meldung und Bedeutung
13	**OUT OF DATA** (nicht genug Daten) Es wird versucht, mit der READ–Anweisung mehr Daten zu lesen, als in DATA–Zeilen deklariert sind.
16	**OUT OF MEMORY** (Speicherüberlauf) Entweder reicht der Speicher für das Programm nicht aus, oder der Stapelspeicher ist wegen zuvieler aktiver DO–, FOR– oder GOSUB–Anweisungen übergelaufen.
15	**OVERFLOW** (Überlauf) Das Ergebnis einer Berechnung übersteigt den größten darstellbaren Wert (1.701411833E+38).
19	**REDIM'D ARRAY** (mehrfache Felddimensionierung) Feldvariablen dürfen in BASIC nur ein einziges Mal dimensioniert werden.
12	**RETURN WITHOUT GOSUB** (RETURN ohne GOSUB) Der Interpreter findet eine RETURN–Anweisung, zu der keine vorausgegangene GOSUB–Anweisung existiert.
23	**STRING TOO LONG** (zu lange Zeichenkette) Zeichenketten dürfen beim Commodore–BASIC nur maximal 255 Zeichen lang sein.
11	**SYNTAX** (Syntaxfehler) Ein BASIC–Befehl, eine BASIC–Anweisung, –Funktion oder –Systemvariable ist falsch geschrieben oder es fehlt eine Klammer, ein Komma o.ä.
1	**TOO MANY FILES** (zu viele offene Dateien) Im Commodore–BASIC sind maximal 10 gleichzeitig geöffnete Dateien erlaubt.
22	**TYPE MISMATCH** (fehlende Variablentyp–Übereinstimmung) Es wird eine numerische Variable verwendet, wo nur Zeichenkettenvariablen erlaubt sind oder umgekehrt.
17	**UNDEF'D STATEMENT** (nichtdefinierte Zeilennummer) Es wird auf eine Zeilennummer Bezug genommen, die in dem laufenden Programm nicht existiert.

BASIC–Fehlermeldungen (Fortsetzung)

Code Meldung und Bedeutung

27 **UNDEFINED FUNCTION** (nichtdefinierte Funktion) Eine anwenderspezifische Funktion wird verwendet, die vorher nicht mit DEF FN definiert wurde.

28 **VERFIY** (Programm–Verifizierungsfehler) Ein mit Hilfe des VERIFY-Befehls ausgeführter Vergleich des Programms im Hauptspeicher mit dem entsprechenden Programm auf Kassette oder Diskette ist nicht erfolgreich.

BASIC–Fehlermeldungen, geordnet nach Fehlercodes

Code	Meldung	Code	Meldung
1	TOO MANY FILES	19	REDIM'D ARRAY
2	FILE OPEN	20	DIVISION BY ZERO
3	FILE NOT OPEN	21	ILLEGAL DIRECT
4	FILE NOT FOUND	22	TYPE MISMATCH
5	DEVICE NOT PRESENT	23	STRING TOO LONG
6	NOT INPUT FILE	24	FILE DATA
7	NOT OUTPUT FILE	25	FORMULA TOO COMPLEX
8	MISSING FILE NAME	26	CAN'T CONTINUE
9	ILLEGAL DEVICE NUMBER	27	UNDEFINED FUNCTION
10	NEXT WITHOUT FOR	28	VERIFY
11	SYNTAX	29	LOAD
12	RETURN WITHOUT GOSUB	30	BREAK
13	OUT OF DATA	31	CAN'T RESUME
14	ILLEGAL QUANTITY	32	LOOP NOT FOUND
15	OVERFLOW	33	LOOP WITHOUT DO
16	OUT OF MEMORY	34	DIRECT MODE ONLY
17	UNDEF'D STATEMENT	35	NO GRAPHICS AREA
18	BAD SUBSCRIPT	36	BAD DISK

8.2 Floppy–Disk–Fehlermeldungen

In diesem Abschnitt werden die Fehlermeldungen des Floppy–Disk–Betriebssystems (DOS) aufgeführt und beschrieben. Der Text der Fehlermeldung sowie der Fehlercode können im C128-Modus mit Hilfe der der Systemvariablen DS$ und DS abgefragt und angezeigt werden. Da der Fehlercode generell Bestandteil des Fehlertextes ist, wird auf eine alphabetische Anordnung verzichtet.

Fehlercodes unter 20 können ignoriert werden. Ausnahme ist der Code 01, der der Meldung über gelöschte Dateien, die vom DOS nach der Ausführung des SCRATCH–Befehl bereitgestellt wird, vorangesetzt ist.

Bei allen Meldungen mit Codes von 20 oder darüber bedeutet die erste Zahl nach der Meldung (***tt***) die Nummer der Disketten–Spur und die zweite Zahl (***ss***) die Nummer des Sektors in dieser Spur, bei der/dem der Fehler aufgetreten ist. Bei der Meldung 01 bedeutet die erste Zahl nach der Meldung die Anzahl der gelöschten Dateien. Die zweite Zahl ist hier immer 00.

Code Meldung und Bedeutung

01, **FILES SCRATCHED,*nn*,00** (Dateien gelöscht) Es wurden *nn* Dateien gelöscht.

20, **READ ERROR,*tt,ss*** (Lesefehler) Das DOS kann für den angegebenen Diskettensektor die Blockkennung nicht identifizieren. Dies kann durch eine nicht erlaubte Sektornummer oder durch eine zerstörte Blockkennung verursacht werden.

21, **READ ERROR,*tt,ss*** (Lesefehler) Das DOS kann für die angegebene Diskettenspur den Synchronisationscode nicht identifizieren. Dies kann durch fehlerhafte Einstellung des Schreib/Lese–Kopfes, eine unformatierte oder eine falsch eingelegte Diskette verursacht werden.

22, **READ ERROR,*tt,ss*** (Lesefehler) Das DOS kann den angegebenen Datenblock nicht lesen oder verifizieren, da dieser nicht korrekt geschrieben wurde. Dieser Fehler tritt im Zusammenhang mit den Blockbefehlen für Direktzugriff auf weist auf einen ungültigen Sektorzugriff hin.

Floppy–Disk–Fehlermeldungen (Fortsetzung)

Code Meldung und Bedeutung

23, **READ ERROR,*tt,ss*** (Lesefehler) Das DOS findet beim Lesen des angegebenen Sektors Fehler in einem oder mehreren Datenbytes durch eine ungültige Prüfsumme. Diese Meldung kann auch auf Erdungsprobleme bei der Floppy–Disk deuten.

24, **READ ERROR,*tt,ss*** (Lesefehler) Das DOS findet beim Lesen der Daten im angegebenen Sektor ungültige Bit–Muster und generiert deshalb einen Hardware–Fehler. Diese Meldung kann auch auf Erdungsprobleme bei der Floppy–Disk deuten.

25, **WRITE ERROR,*tt,ss*** (Schreibfehler) Das DOS findet nach dem Schreiben des angegebenen Sektors keine Übereinstimmung der geschriebenen Daten mit den Daten im Schreibpuffer (read–after–write check).

26, **WRITE PROTECT ON,*tt,ss*** (Schreibschutz aufgeklebt) Das DOS kann auf die Diskette im bezogenen Laufwerk nicht schreiben, weil die Schreibschutzlasche aufgeklebt ist.

27, **READ ERROR,*tt,ss*** (Lesefehler) Das DOS findet in der Blockkennung für den angegebenen Sektor einen Prüfsummenfehler.

28, **WRITE ERROR,*tt,ss*** (Schreibfehler) Das DOS kann den angegeben Sektor wegen Überlänge nicht schreiben.

29, **DISK ID MISMATCH,*tt,ss*** (Keine Diskettenkennungs–Übereinstimmung) Das DOS versucht, auf eine Diskette zuzugreifen, die noch nicht im Laufwerk initialisiert wurde. Dieser Fehler tritt auch bei unleserlicher Diskettenkennung auf.

30, **SYNTAX ERROR,00,00** (Syntaxfehler) Der über den Befehlskanal an die Floppy–Disk übertragene Befehl kann vom DOS nicht interpretiert werden (z.B. fehlerhafte Parameter im Befehl).

31, **SYNTAX ERROR,00,00** (Syntaxfehler) Der an die Floppy–Disk übertragene Befehl ist ungültig. Die Befehlskennung muß das erste Zeichen in der Befehlszeichenkette sein.

Floppy–Disk–Fehlermeldungen (Fortsetzung)

Code Meldung und Bedeutung

32, **SYNTAX ERROR,00,00** (Syntaxfehler) Der an die Floppy–Disk übertragene Befehl ist länger als 58 Zeichen.

33, **SYNTAX ERROR,00,00** (Syntaxfehler) In einer OPEN- oder DOPEN-Anweisung oder bei einem SAVE– oder DSAVE–Befehl wurden verbotenerweise die Jokerzeichen ? oder * (s. Kapitel 6.3) verwendet.

34, **SYNTAX ERROR,00,00** (Syntaxfehler) In einem Befehl oder einer Anweisung fehlt der Dateiname oder kann nicht als solcher identifiziert werden.

39, **SYNTAX ERROR,00,00** (Syntaxfehler) Wie Fehler 31.

50, **RECORD NOT PRESENT,*tt,ss*** (logischer Datensatz nicht vorhanden) Entweder wird versucht, mit GET# oder INPUT# über das logische Ende einer Datei hinaus zu lesen, oder es wird versucht, den Lesezeiger in einer Relativdatei mit der RECORD–Anweisung hinter das logische Ende der Datei zu positionieren. Wenn jedoch damit die Datei in Verbindung mit einer anschließenden PRINT#–Anweisung erweitert werden soll, kann diese Meldung ignoriert werden. Nach dieser Meldung sollten ohne vorherige Neupositionierungen die Anweisungen GET# oder INPUT# nicht verwendet werden.

51, **OVERFLOW IN RECORD,*tt,ss*** (zu langer Satz) Es wird versucht, mit PRINT# einen logischen Datensatz in eine Relativdatei zu schreiben, dessen Länge einschließlich des abschließenden Wagenrücklaufcodes die in der DOPEN–Anweisung angegebene Satzlänge überschreitet.

52, **FILE TOO LARGE,*tt,ss*** (Datei zu groß) Bei der Ausführung einer RECORD–Anweisung zur Dateierweiterung diagnostiziert DOS nicht mehr ausreichend Speicherplatz auf der Diskette.

60, **WRITE FILE OPEN,*tt,ss*** (Ausgabedatei offen) Eine nicht korrekt geschlossene Ausgabedatei kann nicht zum Lesen geöffnet werden.

Floppy–Disk–Fehlermeldungen (Fortsetzung)

Code Meldung und Bedeutung

61, **FILE NOT OPEN,*tt,ss*** (Datei nicht göffnet) Es wird versucht, auf eine nicht geöffnete Datei zuzugreifen. Gelegentlich wird diese Meldung nicht generiert und der Dateizugriff vielmehr von DOS einfach ignoriert.

62, **FILE NOT FOUND,*tt,ss*** (Datei nicht gefunden) Auf der angegebenen Diskette findet das DOS die angegebene Datei nicht.

63, **FILE EXISTS,*tt,ss*** (Datei existiert bereits) Es wird versucht, auf der Diskette im angegebenen Laufwerk eine Ausgabedatei unter einem Namen einzurichten, der bereits für eine existierende Datei verwendet wurde.

64, **FILE TYPE MISMATCH,*tt,ss*** (keine Dateityp–Übereinstimmung) Es wird versucht, auf der Diskette im angegebenen Laufwerk eine existierende Datei unter einem Dateityp zu eröffnen, der nicht mit dem auf der Diskette vermerkten Typ übereinstimmt.

65, **NO BLOCK,*tt,ss*** (Block bereits belegt) Es wird versucht, mit dem DOS–Befehl B–A (block allocate) einen bereits reservierten Block zu reservieren. Bei dieser Meldung geben ***tt*** und ***ss*** den nächst höheren freien Block auf der Diskette an. Sind beide 00, dann sind alle Blöcke mit höheren Disk–Adressen belegt.

66, **ILLEGAL TRACK AND SECTOR,*tt,ss*** (Spur und Sektor ungültig) DOS hat versucht, auf eine Spur oder einen Sektor zuzugreifen, die nicht mit dem aktuellen Format übereinstimmen. Dies kann auf Probleme beim Lesen des Disk–Adreßzeigers hindeuten.

67, **ILLEGAL SYSTEM T OR S,*tt,ss*** (Systemspur oder Systemsektor ungültig) Das DOS diagnostiziert einen Formatfehler bei einer Systemspur oder einem Systemsektor.

70, **NO CHANNEL,*tt,ss*** (Kein Datenpuffer frei) Beim Öffnen einer Datei kann DOS keinen Datenpuffer mehr zuordnen. Zur gleichen Zeit kann das DOS nur maximal 5 offene sequentielle, 3 offene Relativ–Dateien oder 6 Direkzugriffskanäle verwaltet.

Floppy–Disk–Fehlermeldungen (Fortsetzung)

Code Meldung und Bedeutung

71, **DIRECTORY ERROR,*tt,ss*** (Fehler im Diskettenverzeichnis) Die auf der angegebenen Diskette gespeicherte BAM (block availability map = Verzeichnis der belegten und freien Sektoren) stimmt nicht mit der vom DOS im Arbeitspeicher für diese Diskette verwalteten BAM überein. Die Diskette sollte mit

OPEN 15,8,15,"I"

erneut initialisiert werden. Geöffnete Dateien können dadurch möglicherweise geschlossen werden.

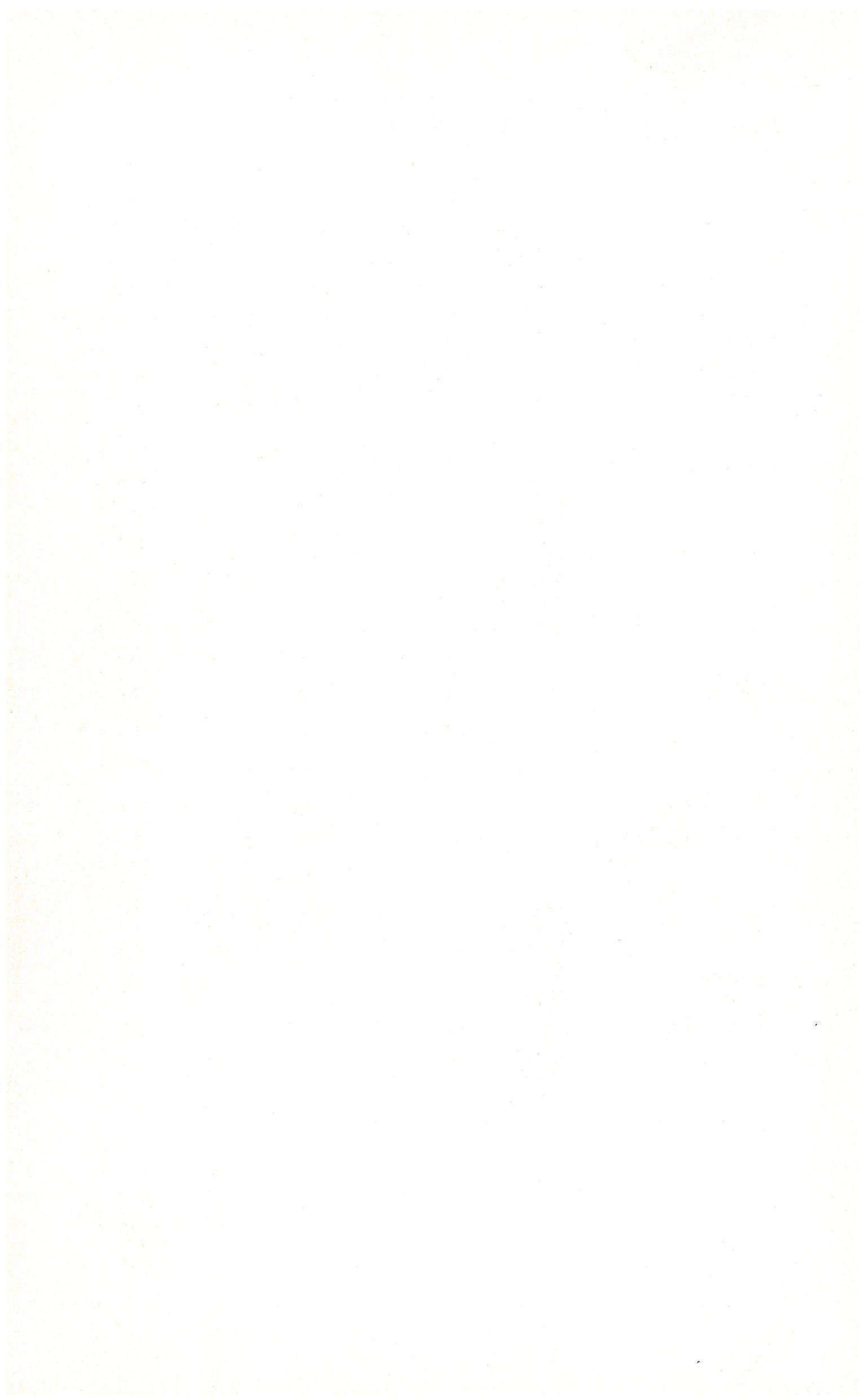
72, **DISK FULL,*tt,ss*** (Diskette voll) Entweder sind alle verfügbaren Sektoren auf der angegebenen Diskette belegt oder das Verzeichnis hat keinen Platz mehr für weitere Einträge (144 maximal).

73, **DOS MISMATCH,*tt,ss*** (falsche DOS–Version) Die Versionen 1 und 2 des DOS sind lese– aber nicht schreibkompatibel. Diese Meldung wird immer dann generiert, wenn versucht wird, auf eine Diskette zu schreiben, die mit einem DOS einer anderen Version formatiert wurde. Außerdem kann diese Meldung nach dem Einschalten angezeigt werden.

74, **DRIVE NOT READY,*tt,ss*** (Laufwerk nicht bereit) Es wird versucht, auf ein Laufwerk ohne eingelegte Disketten zuzugreifen.

Anhang A:

Zeichencode-Tabellen, Steuercodes



ASCII UND CHR$ CODES

Dieser Anhang zeigt für alle x, welches Zeichen auf dem Bildschirm erscheint, wenn Sie PRINT CHR$(x) senden. Umgekehrt können Sie auch das Ergebnis für PRINT ASC(„x") entnehmen, wobei „x" der zugehörigen gedrückten Taste entspricht. Dies ist besonders nützlich, um das in einem GET Statement empfangene Zeichen zu ermitteln, zwischen Groß- und Kleinschrift umzuschalten und um nicht druckbare Steuerzeichen, wie etwa Umschaltung Groß-/Kleinschrift zu senden, die nicht in einem String zwischen Anführungszeichen eingeschlossen werden können.

Zeichen	CHR$	Zeichen	CHR$	Zeichen	CHR$	Zeichen	CHR$
	0	CRSR ↓	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
blockiert SHIFT C=	8		25	*	42	;	59
entriegelt SHIFT C=	9		26	+	43	<	60
	10		27	,	44	=	61
	11	RED	28	–	45	>	62
	12	CRSR →	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
UMSCHALTUNG KLEINBUCHST.	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

Zeichen	CHR$	Zeichen	CHR$	Zeichen	CHR$	Zeichen	CHR$
D	68		97		126		155
E	69		98		127	PUR	156
F	70		99		128	CRSR	157
G	71		100		129	YEL	158
H	72		101		130	CYN	159
I	73		102		131	SPACE	160
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165
O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84		113	UMSCHALTUNG GROSSBUCHST.	142		171
U	85		114		143		172
V	86		115	BLK	144		173
W	87		116	CRSR	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90		119	INST DEL	148		177
[	91		120		149		178
£	92		121		150		179
]	93		122		151		180
↑	94		123		152		181
←	95		124		153		182
	96		125		154		183

Zeichen	CHR$	Zeichen	CHR$	Zeichen	CHR$	Zeichen	CHR$
	184		186		188		190
	185		187		189		191

Codes	**192-233**	**wie Codes**	**96-127**
Codes	**224-254**	**wie Codes**	**160-190**
Code	**255**	**wie Code**	**126**

BILDSCHIRM CODES (40-Spalten-Bildschirm)

Die folgende Tabelle listet alle in den Zeichensätzen des COMMODORE 128 vorkommenden Zeichen auf. Sie zeigt, welche Zahl in den Bildschirmspeicher (Adressen 1024-2023) gePOKEd werden muß, um ein bestimmtes Zeichen zu erhalten. Sie zeigt umgekehrt, welches Zeichen einer aus dem Bildschirmspeicher gePEEKten Zahl entspricht.

Zwei Zeichensätze sind verfügbar, doch immer nur einer zu einer Zeit. Das heißt, Sie können nicht Zeichen eines Satzes auf den Bildschirm bringen, während auch Zeichen des anderen Satzes angezeigt werden. Zwischen den Zeichensätzen wird umgeschaltet, indem man die SHIFT-Taste und die COMMODORE-Taste gleichzeitig drückt.

Von Basic können Sie mit PRINT CHR$ (142) in den Groß-/Grafik-Modus und mit PRINT CHR$ (14) in den Groß-/Klein-Modus umschalten.

Jedes Zeichen der Tabelle kann auch in REVERS (dunkel auf hellem Grund) dargestellt werden, indem Sie 128 zum angegebenen Wert addieren.

Wollen Sie zum Beispiel in 1504 einen ausgefüllten Kreis darstellen, POKEn Sie in diese Adresse den Code (81) des ausgefüllten Kreises: POKE 1504,81.

Zu jedem Platz des Bildschirmspeichers gibt es eine korrespondierende Adresse (55296-56295), die die Farbe des Zeichens kontrolliert. Um die Farbe des Kreises in gelb (Farbcode 7) zu ändern, POKEn Sie die entsprechende Adresse (55776) mit dem Farbwert: POKE 55776,7.

In Anhang B finden Sie die vollständigen Adressätze des Bildschirm- und Farbspeichers, zusammen mit den Farbcodes.

BILDSCHIRM CODES

Satz 1	Satz 2	Poke	Satz 1	Satz 2	Poke	Satz 1	Satz 2	Poke
@		0	V	v	22	,		44
A	a	1	W	w	23	–		45
B	b	2	X	x	24	.		46
C	c	3	Y	y	25	/		47
D	d	4	Z	z	26	0		48
E	e	5	[		27	1		49
F	f	6	£		28	2		50
G	g	7	]		29	3		51
H	h	8	↑		30	4		52
I	i	9	←		31	5		53
J	j	10	SPACE		32	6		54
K	k	11	!		33	7		55
L	l	12	"		34	8		56
M	m	13	#		35	9		57
N	n	14	$		36	:		58
O	o	15	%		37	;		59
P	p	16	&		38	<		60
Q	q	17	'		39	=		61
R	r	18	(		40	>		62
S	s	19	)		41	?		63
T	t	20	*		42			
U	u	21	+		43			

Satz 1	Satz 2	Poke
		64
	A	65
	B	66
	C	67
	D	68
	E	69
	F	70
	G	71
	H	72
	I	73
	J	74
	K	75
	L	76
	M	77
	N	78
	O	79
	P	80
	Q	81
	R	82
	S	83
	T	84
	U	85
	V	86
	W	87
	X	88
	Y	89
	Z	90
		91
		92
		93
		94
		95
SPACE		96
		97
		98
		99
		100
		101
		102
		103
		104
		105
		106
		107
		108
		109
		110
		111
		112
		113
		114
		115
		116
		117
		118
		119
		120
		121
		122
		123
		124
		125
		126
		127

Die Codes 128-255 ergeben die invers dargestellten Zeichen der Codes 0-127.

An den freien Stellen in Spalte 2 stimmen die beiden Sätze überein.

COMMODORE 128 CODETABELLE

Dez	Hex	Funktion
0		
1		
2	02	Unterstreichen ein (80 Zeichen-Bildschirm)
3		
4		
5	05	Farbe weiß
6		
7	07	Klingelzeichen
8		
9	09	Tabulator
10	0a	Zeilenvorschub
11	0b	blockiert Umschaltung Text/ Grafik-Zeichensatz
12	0c	entriegelt Umschaltung Text/ Grafik-Zeichensatz
13	0d	RETURN (Wagenrücklauf und Zeilenvorschub)
14	0e	Umschaltung auf Text-Zeichensatz
15	0f	Blinken ein (80 Zeichen-Bildschim)
16		
17	11	CRSR abwärts
18	12	revers ein
19	13	home
20	14	delete
21		
22		
23		
24	18	Tabulator setzen/löschen
25		
26		
27	1b	ESCAPE
28		
29		
30		
31		

Dez	Hex	ASCII		DIN	
		Grafik	groß/klein	Grafik	groß/klein
32	20	SPACE	SPACE	SPACE	SPACE
33	21	!	!	!	!
34	22	"	"	"	"
35	23	#	#	#	#
36	24	$	$	$	$
37	25	%	%	%	%
38	26	&	&	&	&
39	27	'	'	'	'
40	28	(	(	(	(
41	29	)	)	)	)
42	2a	*	*	*	*
43	2b	+	+	+	+
44	2c	,	,	,	,
45	2d	−	−	−	−
46	2e	.	.	.	.
47	2f	/	/	/	/
48	30	0	0	0	0
49	31	1	1	1	1
50	32	2	2	2	2
51	33	3	3	3	3
52	34	4	4	4	4
53	35	5	5	5	5
54	36	6	6	6	6
55	37	7	7	7	7
56	38	8	8	8	8
57	39	9	9	9	9
58	3a	:	:	:	:
59	3b	;	;	;	;
60	3c	<	<	<	<
61	3d	=	=	=	=
62	3e	>	>	>	>
63	3f	?	?	?	?
64	40	@	@	§	§
65	41	A	a	A	a
66	42	B	b	B	b
67	43	C	c	C	c
68	44	D	d	D	d

Dez	Hex	ASCII Grafik	ASCII groß/klein	DIN Grafik	DIN groß/klein
69	45	E	e	E	e
70	46	F	f	F	f
71	47	G	g	G	g
72	48	H	h	H	h
73	49	I	i	I	i
74	4a	J	j	J	j
75	4b	K	k	K	k
76	4c	L	l	L	l
77	4d	M	m	M	m
78	4e	N	n	N	n
79	4f	O	o	O	o
80	50	P	p	P	p
81	51	Q	q	Q	q
82	52	R	r	R	r
83	53	S	s	S	s
84	54	T	t	T	t
85	55	U	u	U	u
86	56	V	v	V	v
87	57	W	w	W	w
88	58	X	x	X	x
89	59	Y	y	Y	y
90	5a	Z	z	Z	z
91	5b	[	[	[	[
92	5c	£	£	\	\
93	5d	]	]	]	]
94	5e	↑	↑	↑	↑
95	5f	←	←	—	—
96	60			`	`
97	61	♠	A		A
98	62		B		B
99	63		C		C
100	64		D		D
101	65		E		E
102	66		F		F
103	67		G		G
104	68		H		H
105	69		I		I
106	6a		J		J
107	6b		K		K
108	6c		L		L
109	6d		M		M
110	6e		N		N
111	6f		O		O
112	70		P		P
113	71		Q		Q
114	72		R		R
115	73	♥	S		S
116	74		T		T
117	75		U		U
118	76		V		V
119	77		W		W
120	78	♣	X		X
121	79		Y		Y
122	7a	♦	Z		Z
123	7b				Ä
124	7c				Ö
125	7d				Ü
126	7e			π	π
127	7f				

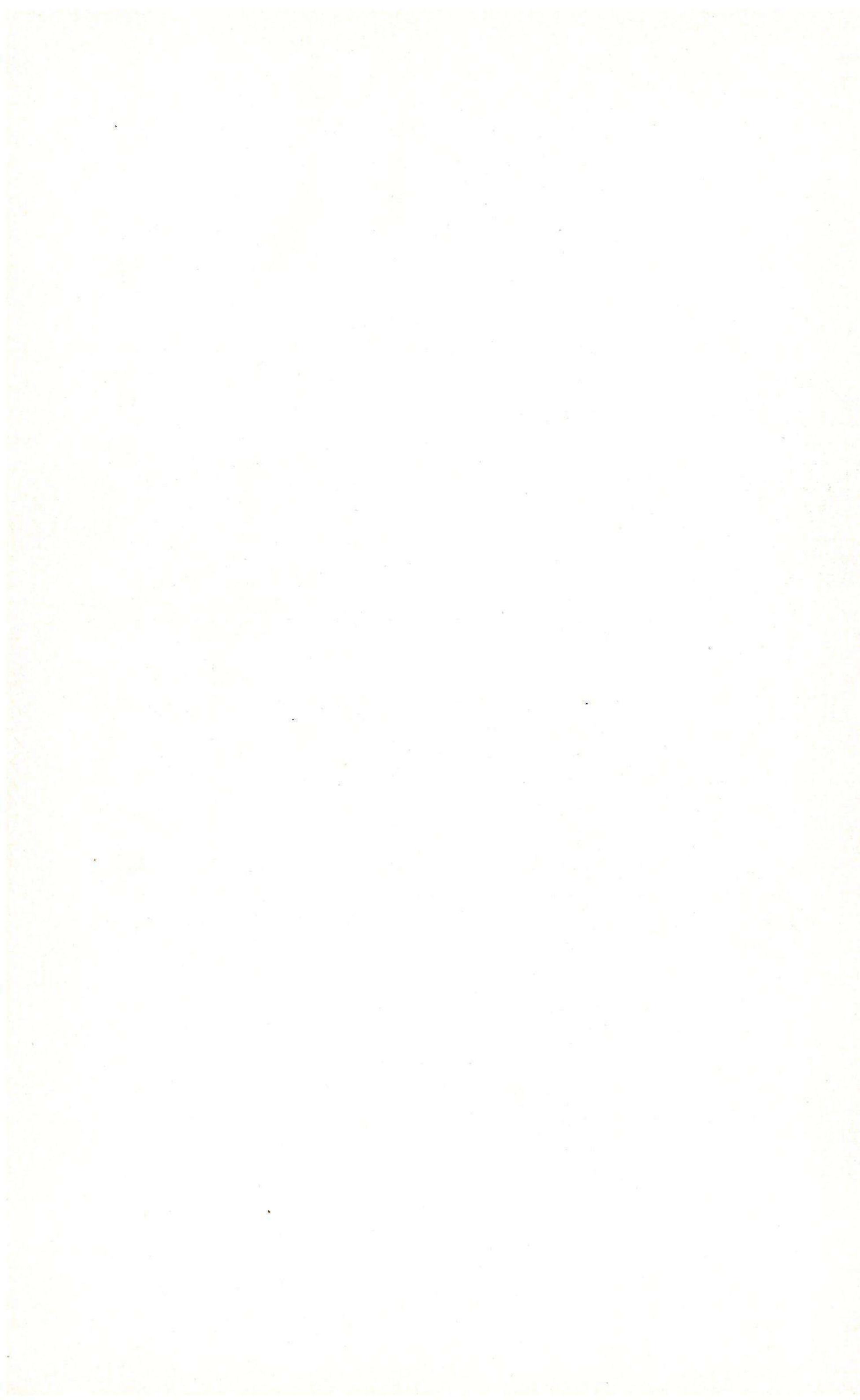
Dez	Hex	Funktion
128	80	
129	81	orange
130	82	Unterstreichen aus
131		
132	84	HELP
133	85	f1
134	86	f3
135	87	f5
136	88	f7
137	89	f2
138	8a	f4
139	8b	f6
140	8c	f8
141	8d	SHIFT RETURN
142	8e	Umschaltung auf Grafik-Zeichensatz
143	8f	Blinken aus
144	90	Schwarz
145	91	Cursor hoch
146	92	Revers aus
147	93	CLEAR
148	94	INST
149	95	braun
150	96	hellrot
151	97	grau 1
152	98	grau 2
153	99	hellgrün
154	9a	hellblau
155	9b	grau 3
156	9c	purpur
157	9d	Cursor links
158	9e	gelb
159	9f	türkis

Dez	Hex	ASCII Grafik	ASCII groß/klein	DIN Grafik	DIN groß/klein
160	a0	SHIFT-Space		SHIFT-Space	
161	a1				
162	a2				
163	a3				
164	a4				
165	a5				
166	a6				
167	a7				
168	a8				
169	a9				
170	aa				
171	ab				
172	ac			é	é
173	ad			£	£
174	ae			è	è
175	af			´	´
176	b0			@	@
177	b1			μ	μ
178	b2			à	à
179	b3			ù	ù
180	b4			â	â
181	b5			ê	ê
182	b6			î	î
183	b7			ô	ô
184	b8			û	û
185	b9			√	√
186	ba		✓	Σ	Σ
187	bb			Ä	ä
188	bc			Ö	ö
189	bd			Ü	ü
190	be			ß	ß
191	bf			^	^
192	c0			`	`
193	c1		A		A
194	c2		B		B
195	c3		C		C
196	c4		D		D
197	c5		E		E
198	c6		F		F
199	c7		G		G
200	c8		H		H
201	c9		I		I

Dez	Hex	ASCII Grafik	ASCII groß/klein	DIN Grafik	DIN groß/klein
202	ca		J		J
203	cb		K		K
204	cc		L		L
205	cd		M		M
206	ce		N		N
207	cf		O		O
208	d0		P		P
209	d1		Q		Q
210	d2		R		R
211	d3		S		S
212	d4		T		T
213	d5		U		U
214	d6		V		V
215	d7		W		W
216	d8		X		X
217	d9		Y		Y
218	da		Z		Z
219	db				Ä
220	dc				Ö
221	dd				Ü
222	de			π	π
223	df			–	–
224	e0	SHIFT-Space		SHIFT-Space	
225	e1				
226	e2				
227	e3				
228	e4				
229	e5				
230	e6				
231	e7				
232	e8				
233	e9				
234	ea				
235	eb				
236	ec			é	é
237	ed			£	£
238	ee			è	è
239	ef			'	'
240	f0			@	@
241	f1			µ	µ
242	f2			à	à
243	f3			ù	ù
244	f4			â	â
245	f5			ê	ê
246	f6			↑	↑
247	f7			ô	ô
248	f8			û	û
249	f9			√	√
250	fa		✓	Σ	Σ
251	fb			Ä	ä
252	fc			Ö	ö
253	fd			Ü	ü
254	fe			ß	ß
255	ff	π		π	π

Anhang B:

Speicherorganisation



Anhang B: Die Speicherverwaltung mit der MMU

B.1.1 Einführung

Die Speicher–Verwaltungseinheit (Memory Management Unit – MMU) wurde entworfen, um die komplexen Vorgänge im C128–System zu kontollieren. Sie wickelt alle C64– Operationen im C64–Modus ab, ist also absolut kompatibel zum C64. Zusätzlich kontrolliert die MMU die Organisation spezieller C128–Operationen einschließlich des CP/M–Modus (Z80–Prozessor).

Hier eine kurze Zusammenfassung der MMU–Merkmale:

1. Bereitstellung und Verwaltung eines umgesetzten Adreßbusses (Translated–Adress–Bus TA8 – TA15).

2. Bereitstellung und Verwaltung der drei verschiedenen Betriebsmodi (C128 – C64 – CP/M).

3. Bereitstellung und Verwaltung der CAS–Auswahl–Leitungen für die RAM–Speicherbankumschaltung.

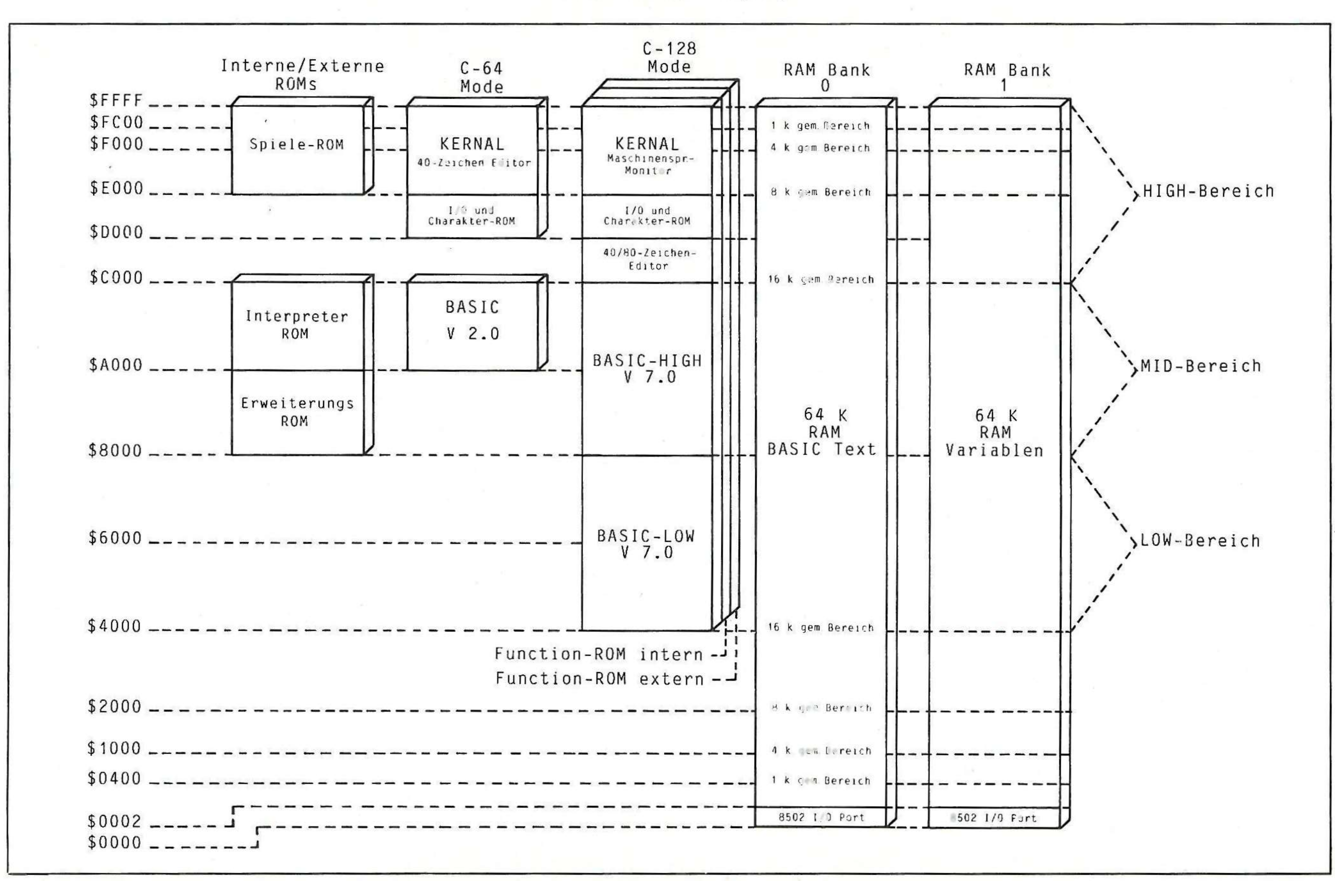
B.1.2 C128–Speicherorganisation

Die Speicherorganisation wird durch spezielle Register der MMU kontrolliert. Diese Register steuern das Umsetzen der Adressen des 8502–Prozessors. Dabei werden 64 kByte Adreßraum in bis zu 1 MByte RAM umgesetzt und verwaltet.

Weiterhin sind bis zu 96 kBytes interner ROM und 32 kBytes externer ROM im C128–System verfügbar.

In Bild B.1 auf der folgenden Seite ist die Speicherorganisation des C128 für den C64– und den C128–Modus zusammengefaßt.

C 128 Memory Belegung



B.1.3 C128–ROM–Organisation

Der C128–Modus ist immer nach dem Einschalten oder nach einem Systemreset aktiv. Dieser Modus wird durch ein Bit im MMU–Konfigurations–Register kontrolliert. Im C128–Modus beansprucht die MMU den Speicherbereich ab Adresse $FF00 und im Ein/Ausgabe (I/O)–Bereich den Platz ab Adresse $D500. Die Benutzung der MMU–Register, die ab $FF00 angelegt sind, erlaubt die Speicherverwaltung ohne Beanspruchung der Register im I/O–Bereich und gewährleistet damit ein Minimum an Speicherplatzverlust. Im C64–Modus belegen die MMU–Register keinen Speicherplatz, obwohl die MMU immer noch von der Hardware für die Speicherverwaltung benutzt wird.

Die ROMs im C64–Modus entsprechen denen des Commodore 64. Das interne BASIC und das Betriebssystem des C128 versorgen den C64–Modus mit dem normalen Commodore 64 Betriebssystem. Dazu wird ein Teil des ROMs, der im C128–Modus verwendet wird, dupliziert.

Im C128–Modus stehen bis zu 48 kBytes Systemsoftware zur Verfügung. Wieviel davon genutzt wird, wird durch geeignete Steuer–Software bestimmt. Da nicht benötigte Teile des Betriebssystems einfach weggeschaltet werden, ist ein schnellerer Zugriff auf den dahinterliegenden RAM möglich.

Die in der Speicheraufteilung nach Bild B.1 erwähnten externen ROMs werden im C64–Modus verwendet und gehorchen den C64–Regeln in bezug auf Speicherumschaltung, d.h. Steckmodule schalten sich selbst über die EXROM– und GAME–Steuerleitungen ein.

Externe ROMs im C128–Modus gehorchen den Regeln ähnlich dem TED-schema für ROM–Umschaltung, d.h. sie werden bei der Systeminitialisierung abgefragt, um zu prüfen, ob sie vorhanden sind und mit welcher Priorität sie zugeschaltet werden.

Diese Methode ist weitaus flexibler als eine verdrahtete ROM–Austauschmethode, denn Betriebssystem– und BASIC–ROM können von einem Anwenderprogramm gegen anwenderspezifische ROMs getauscht oder auch ganz einfach weggeschaltet werden. Eine solche Speichermanipulation wird durch Beschreiben des MMU–Konfigurations–Registers bei der Adresse $D500 oder $FF00 erreicht.

Die Hardware hat die Fähigkeit, die vorher eingegebenen Werte für eine bestimmte Konfiguration zu speichern und erzwingt das Laden des Konfigurations–Registers, in dem eines der Lade–Konfigurations–Register (LCR)

beschrieben wird. Dies erlaubt dem Programmierer z.B. dadurch jederzeit Zugang zu einer RAM–Speicherbank zu haben, daß ein ROM in dieser Speicherbank durch Voreinstellung nicht eingeschaltet wird.

B.1.4 RAM–Speicherorganisation

Wir beziehen uns nochmals auf das Bild B.1 (C128–Speicherorganisation). Für der vorhandenen RAM–Bereich von 128 kBytes werden dynamische Speicherbausteine verwendet. Dieser RAM–Bereich wird in zwei Speicherbänke von je 64 kBytes eingeteilt (64 kBytes ist der maximale Adreßraum, den die 8502– und Z80–Prozessoren adressieren können). Zwischen den Speicherbänken kann umgeschaltet werden.

Der als RAM gekennzeichnete Bereich ist der Speicher, den der Mikroprozessor "sieht", wenn alle ROMs weggeschaltet sind.

Die Speicherbankumschaltung kann auf zwei verschiedene Weisen vorgenommen werden.

Die aktuelle Speicherbank ergibt sich aus dem Wert, der im Konfigurations-Register gespeichert ist. (Es gibt noch andere Speicher–Modi, die sich über die gerade eingeschaltete Bank hinwegsetzen. Diese Modi werden in einem späteren Abschnitt behandelt.) Eine Änderung dieses Registers wirkt sich immer sofort und direkt aus. Ein indirektes Benutzen dieser Register mit programmierten Bank–Konfigurationswerten besteht im Beschreiben eines der indirekten Lade–Register (LCR – Load Configuration Register).

Die LCR–Register sind im Speicherbereich ab der Adresse $FF00 angelegt. Durch Beschreiben eines der LCRs wird der Inhalt des mit ihm korrespondierenden PCRs (Prä–Konfigurations–Register) in das Konfigurations–Register übertragen (es wird in das Konfigurations–Register geodert). Auf diese Weise kann der Programmierer bis zu 4 verschiedene vorprogammierte Zustände einstellen.

So könnte z.B. Bank 1 eine Datenbank sein, also ein reiner RAM–Bereich ohne zugeschalteten ROM oder Ein–/Ausgabe–Bereich, während in Bank 0 das System–ROM und der Ein–/Ausgabebereich durch Voreinstellung eingeschaltet sind.

Wenn man in einer 64–kBytes–Bank arbeitet, ist es manchmal wünschenswert, Daten in der Bank 1 zu hinterlegen, aber dennoch das System–RAM

(Stack, Zero Page, Bildschirm) zu behalten. Für diesen Zweck ist die MMU in der Lage, einen gemeinsamen RAM–Bereich bereitzustellen.

Dieser RAM–Bereich kann zu– oder abgeschaltet werden. Er ist in seiner Größe veränderbar und kann programmiert am Anfang oder am Ende einer Bank angelegt werden. Der gemeinsame RAM–Bereich kann aber auch aufgeteilt werden und ein Teil kann am Anfang und der andere Teil am Ende der Bank stehen.

Die Größe eines gemeisamen RAM–Bereiches wird durch die Bits 0 und 1 im RAM–Konfigurations–Register (RCR) bestimmt. Wenn beide Bits = 0 sind, sind 1 kByte, ist Bit 0 = 1 und Bit 1 = 0 sind 4 kByte, ist Bit 0 = 0 und Bit 1 = 1 sind 8 kByte und sind beide Bit = 1, sind 16 kByte als gemeinsamer Bereich eingeschaltet.

Ist Bit 2 des RCRs gesetzt, wird der gemeinsame Bereich am unteren Ende des RAM–Bereiches, ist Bit 3 im RCR gesetzt, am oberen Ende des RAM–Bereiches angelegt. In allen Fällen ist der gemeinsame Speicherbereich physikalisch ein Bestandteil der Speicherbank 0.

Die ersten 512 Bytes des Speichers (Zero Page (P0) und Stapelspeicher (P1)) können unabhängig vom RCR im gesamten Adreßraum angeordnet werden. Wenn der Prozessor eine Adresse der Zero Page oder des Stapelspeichers anwählt, addiert die MMU den Inhalt des P0– oder des P1–Registerpaares auf die Adesse und legt die neue, erweiterte Adresse (Adressenbits A16 und A17) auf den Bus. Dabei kann natürlich eine Speicherumschaltung (RAM Banking) vorkommen. Wenn die P0– bzw. P1–Register beschrieben werden, werden sie solange zwischengespeichert bis das niederwertige Byte im P0–Register beschrieben wird. Dadurch werden vorzeitige Änderungen der Systemkonfiguration oder ungültige Systemzustände vermieden.

Gleichzeitig werden Zero Page und Stapelspeicher mit dem Speicher vertauscht, der beide ersetzt hat, wenn der Prozessor eine Adresse verwendet, die innerhalb des durch die P0– oder P1–Register bezeichneten Bereiches liegt.

Es ist jedoch zu beachten, daß bei einem System–Reset die Zeiger auf die tatsächliche Zero Page und den tatsächlichen Stapelspeicher gesetzt werden.

Der VIC–Baustein kann durch 2 Bits im MMU–RAM–Konfigurations–Register adessiert werden. Das Register steuert die Adressleitungen A16 und A17 und erlaubt damit, den VIC–Baustein beliebig innerhalb eines 256–kBytes-Bereiches zu plazieren.

Anmerkung: Die Address–Enable–Control (AEC)–Leitung wird von der MMU benutzt, um den VIC–Adreßraum zu steuern, d.h., wenn AEC = 0 ist, kann der VIC–Baustein adressiert werden.

Dies bedeutet, daß ein von außen gesteuerter direkter Speicherzugriff (DMA), der auch auf die AEC–Leitung einwirkt und die Leitung ebenfalls nach logisch Null zieht, den VIC–Baustein ansprechen kann.

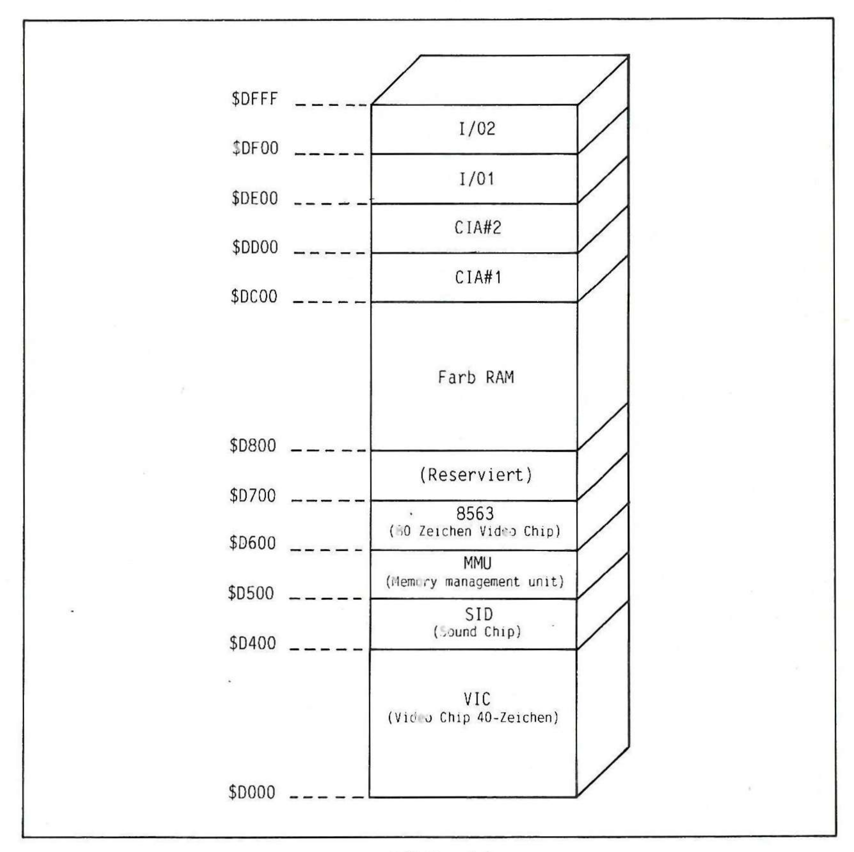
B.1.5 Anordnung von MMU und Ein–/Ausgabebereich im Speicher

Der in Bild B.2 gezeigte Ein–/Ausgabe–Speicherbereich (I/O–Bereich) ist eine vergrößerte Darstellung des in Bild B.1 als "I/O und Character ROM" gekennzeichneten Bereiches.

Wenn der I/O–Bereich zugeschaltet ist, können VIC, SID, alle sonstigen Ein–/Ausgabeeinrichtungen sowie die zusätzlich eingefügten MMU–Register angesprochen werden. Alle I/O–Funktionen sind an der Stelle verblieben, wo sie beim C64 waren mit Ausnahme der MMU, die hinzugefügt wurde.

Mit Ausnahme von 4 Registern, die beim C128–Modus in der Zero Page angeordnet wurden, sind die neuen MMU–Register bei ungenutzten Adressen des I/O–Bereichs angelegt.

Adressen der I/O Bausteine



I/0 Bereich

B.2 Beschreibung der MMU–Register

Bild B.3 zeigt die Register der MMU zusammen mit Ihren Adressen. Im folgenden werden die einzelnen Register beschrieben.

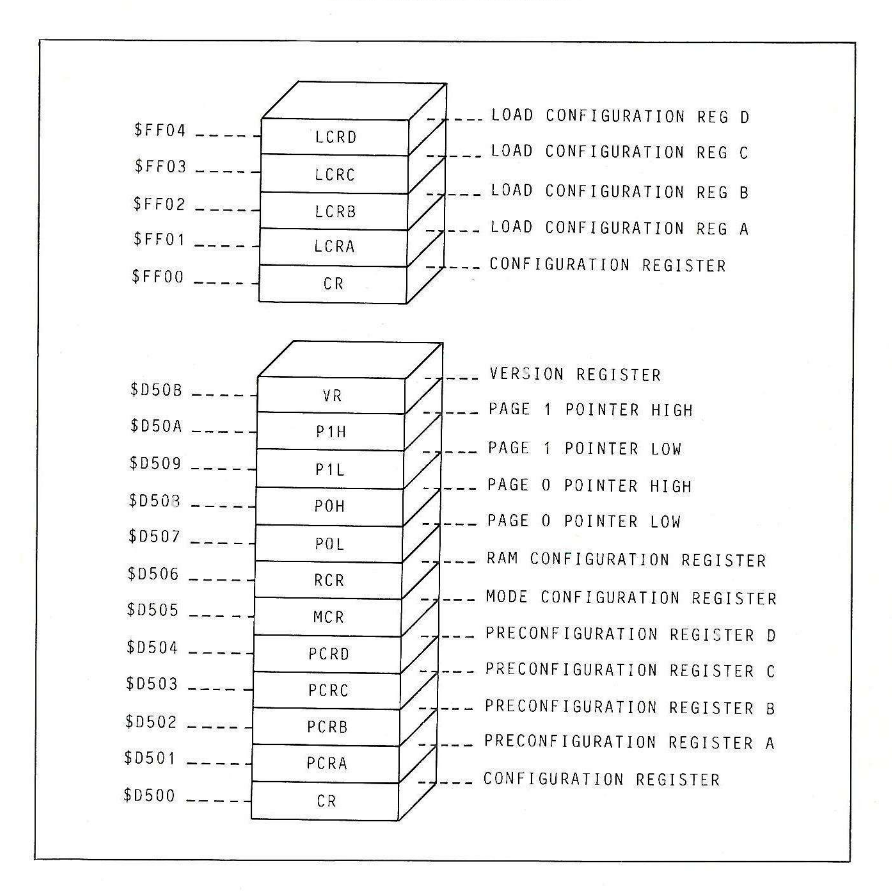
Zu beachten ist, daß die MMU im C64–Modus weggeschaltet ist und die Daten–Ein– und –Ausgabe nur dann gültig ist, wenn die AEC–Leitung logisch "hoch" (1) liegt. Dies ist notwendig, um Buskonflikte während eines VIC–Zyklusses zu vermeiden.

B.2.1 Das Konfigurations–Register

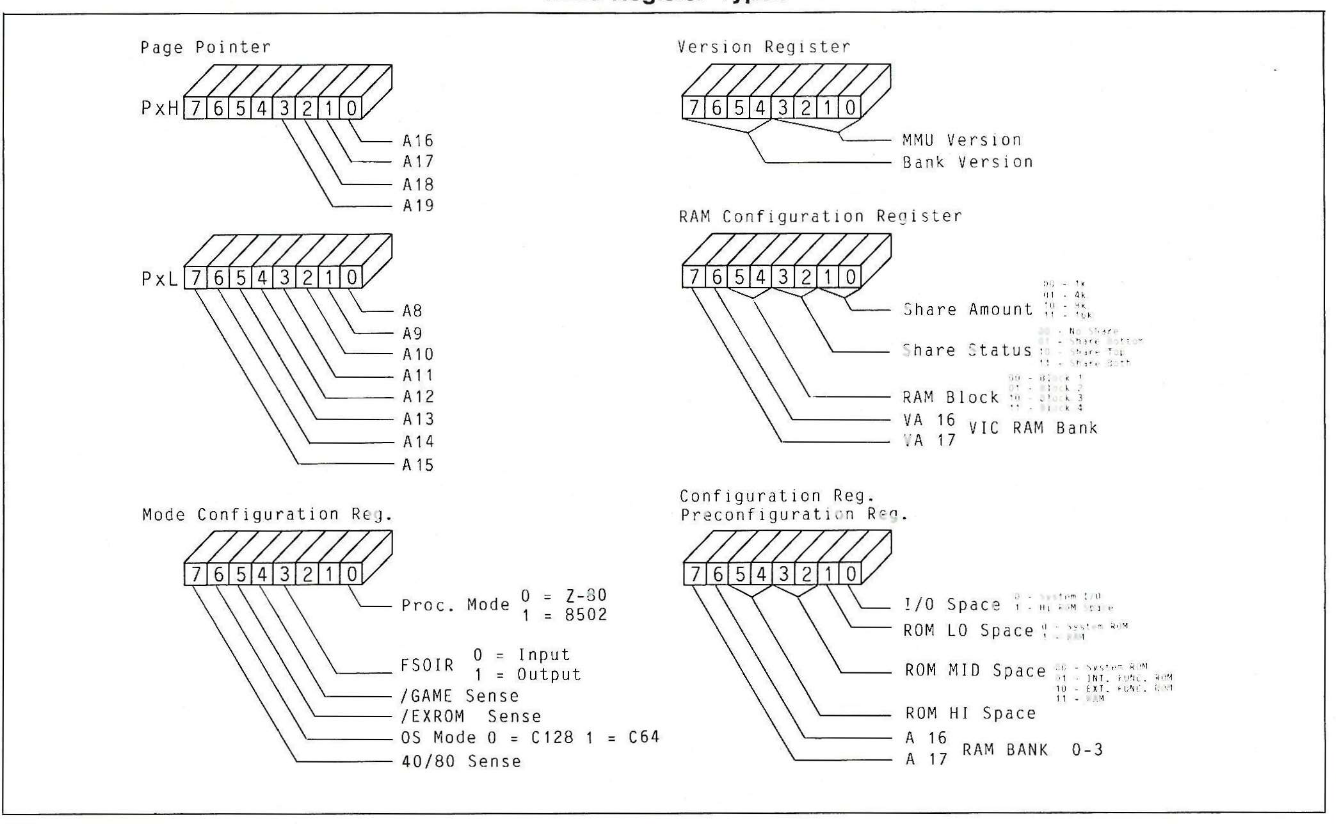
Das Konfigurations–Register (CR) kontrolliert ROM–, RAM– und I/O–Konfiguration im C128–System. Es liegt sowohl bei Adresse $D500 des I/O–Bereiches als auch bei Adresse $FF00.

Im C128–Modus kontrolliert das Bit 0, ob der I/O–Bereich ($D500–$D5FF) eingeschaltet oder ob der ROM/RAM–Bereich zugeschaltet ist. Eine 0 schaltet die I/O–Konfiguration ein. Eine 1 erlaubt eine der verschiedenen ROM/RAM–Kombinationen einzuschalten. Welche, hängt vom Zustand der anderen Bits im Konfigurations–Register ab. Der Wert dieses Bits wird zwischengespeichert, bis der Taktimpuls abfällt, um einen unstabilen Zustand zu vermeiden. Im C64–Modus wird die I/O–Select–Leitung durch dieses Bit auf 1 festgehalten. Zu beachten ist, daß, wenn der I/O–Bereich nicht eingeschaltet ist, der ROM/RAM–Zugriff durch die ROM–High–Konfigurations–Bits kontrolliert wird (dies wird in einem späteren Abschnitt behandelt). Das Bit 0 ist nach dem Einschalten oder nach einem Systemreset auf 0 gesetzt. Wenn Bit 0 = 1 ist, schalten sich die MMU–Register ($D500–$D508) dazu. Wenn Bit 0 = 0 ist, werden die Register aus dem Speicherbereich weggeschaltet. Die MMU–Register im Bereich von $FF00–$FF04 sind im C128–Modus immer präsent. Die I/O–Select–Leitung zeigt im C128–Modus den Zustand dieses Bits an.

MMU Register Belegung



MMU Register Typen



Das Bit 1 kontrolliert im C128–Modus den Prozessorzugriff zum niederwertigen ROM–Bereich (s. Bild B.1), der im Adreßraum $4000–$7FFF angelegt ist. Ist das Bit 1, so erscheint der Bereich als RAM und wird mit CAS–Signalen versorgt. Wenn Bit 1 = 0 ist, wird System–ROM in diesen Bereich eingeschaltet. Dieses Bit hat Auswirkung auf die Speicher–Status–Leitungen MS0 und MS1. Es wird im PLA (Progamable Logic Array) decodiert und gibt den entsprechenden ROM–Steckplatz (chip select) frei. Wurde an dieser Stelle ROM gewählt, werden die beiden MS0– und MS1–Leitungen nach 0 gezogen, wenn der Prozessor diesen Bereich adressiert. Bit 1 wird auf 0 gesetzt, wenn im C128–Modus das BASIC–Low–ROM (s. Bild B.1) eingeschaltet werden soll.

Die nächsten zwei Bits (Bit 2 und 3) bestimmen im C128–Modus die Speicherart (ROM oder RAM) für den mittleren Bereich, der im Adressraum von $8000–$BFFF angelegt ist. Wenn beide Bits auf 0, sind wird hier das System–ROM angelagert. Ist nur das Bit 2 = 1, wird das interne Funktions-RAM zugeschaltet. Das externe Funktions–ROM ist eingeschaltet, wenn das Bit 3 alleine 1 ist. RAM wird in diesem Bereich angelegt, wenn beide Bits auf 1 gesetzt sind.

Diese beiden Bits haben auch eine Auswirkung auf die Speicherstatus–Leitungen:

Wenn der mittlere Bereich adressiert wird, reflektiert MS0 den Zustand von Bit 3 und MS1 den Zustand von Bit 2. Beide Bits werden auf 0 gesetzt, wenn der BASIC–High–ROM eingeschaltet wird.

Die Bits 4 und 5 bestimmen die Speicherart (RAM oder ROM im Adreßraum $C000–$FFFF. Diese beiden Bits haben keinen Einfuß auf den C64–Modus. Ähnlich wie im mittleren Bereich, ist, wenn beide Bits auf 0 gesetzt sind, das System–ROM eingeschaltet.

Ist Bit 4 allein 1, wird das interne Funktions–ROM zugeschaltet. Das externe Funktions–ROM ist eingeschaltet, wenn Bit 5 allein 1 ist. Sind beide Bits auf 1 gesetzt, ist in diesem Bereich RAM angelegt. Zu beachten ist, daß, wenn das I/O–Bit 1 ist, der Bereich zwischen $D500 und $D5FF unabhängig vom Zustand dieser Bits für I/O–Funktionen reserviert ist. Ist das I/O–Bit 0, ist in diesem Bereich ($D000–$DFFF) der Zeichengenerator (Character ROM) angelegt. Das bedeutet, daß im oberen ROM im Bereich $D000–$DFFF immer eine Lücke existiert. Wie bei den anderen ROM–Select–Bits beeinflussen diese beiden Bits auch die Memory–Select–Leitungen. Bit 5 korrespon-

diert mit MS1 und Bit 4 mit MS0. Die beiden Bits werden durch Einschalten oder durch einen System–Reset auf 0 gelegt und die ROMs mit Betriebssystem und Zeichengenerator werden eingeschaltet.

Auch hier ist zu beachten, daß im C128–Modus ebenfalls eine Lücke für die MMU–Register im Bereich von $FF04 vorhanden ist. Diese Lücke wird durch die beiden MS–Leitungen und die beiden CAS–Enable–Leitungen geschaffen, die alle auf 1 gehalten werden.

Die beiden letzten Bits (Bit 6 und 7) kontrollieren die RAM–Bank–Auswahl. Ihre Funktion hängt von der MMU–Version ab. Für die vorliegende Version als 128–kBytes–System ist nur das Bit 6 signifikant. Ist das Bit 6 auf 0, ist die Bank 0 eingeschaltet, indem das CAS0–Signal freigegeben wird. Ist Bit 6 = 1, wird das CAS1–Signal freigegeben und schaltet damit die Bank 1 ein. Bit 7 ist in dieser Version inaktiv.

B.2.2 Der Präkonfigurations–Mechanismus

Der Präkonfigurations–Mechanismus ist eine Besonderheit der MMU. Er erlaubt mit einem Minimum an Zeit– und Speicheraufwand, das Konfigurations–Register mit seinen verschiedenen Konfigurationen zu laden.

Dabei werden zwei Registersätze verwendet, die Präkonfigurations–Register und die Lade–Konfigurations–Register.

Die Präkonfigurations–Register (PCRA – PCRD) werden für die Speicherung mehrerer verschiedener Speicherkonfigurationen benutzt und können mit einem einfachen Befehl geändert werden. Das Format jedes PCRs gleicht dem Format des Konfigurations–Registers (CR). Wird ein Wert in ein PCR gespeichert, hat dies noch keine unmittelbare Auswirkung. Die PCR belegen den I/O–Bereich von $D501– $D504. Diese Register sind nach dem Einschalten oder nach einem System–Reset auf 0 gesetzt.

Die Lade–Konfigurations–Register (LCRA – LCRD) korrespondieren direkt mit den PC–Registern. Wird in ein LCR geschrieben, veranlaßt dieses, daß der Inhalt des mit dem LCR korrespondierenden PCR in das CR gebracht wird. Die LCRs sind im Bereich $FF01–$FF04 angelegt. Weder die LCRs noch die PCRs wirken sich im C64–Modus aus. Zu beachten ist, daß diese Register und das CR bei $FF00 immer verfügbar sind, unabhängig ob ROM, RAM oder die High–ROM–Bankkonfiguration eingeschaltet ist. Jede Adressierung in diesem Bereich veranlaßt die MMU, die beiden Memory–StatusLeitungen sowie die beiden CAS–Enable–Leitungen auf 1 zu legen.

B.2.3 Das Modus–Konfigurations–Register

Der augenblicklich eingeschaltete Rechner–Modus wird durch das Modus-Konfigurations–Register (MCR) bestimmt. Es kontrolliert, welcher Prozessor (8502 oder Z80) das System steuert, welcher der Systemmodi (C128 oder C64) gerade eingeschaltet ist und steuert außerdem verschiedene andere Systembesonderheiten.

Das erste Bit (Bit 0) bestimmt, welcher Prozessor aktiviert ist. Es kann als invertierter Ausgang des Z80–Enable–Signals angesehen werden. Liegt dieses Bit auf 0, zeigt es an, daß der 8502 Prozessor aktiv ist, liegt es auf 1, ist der Z80–Prozessor aktiv. Dieses Bit wird nach dem Einschalten oder nach einem Systemreset auf 0 eingestellt.

Alle Zugriffe auf den Speicherbereich $0000–$0FFF werden in den Speicherbereich $D000–$DFFF übertragen, genau an die Stelle, wo das CP/M–ROM–BIOS physikalisch angelegt ist.

Um zu verhindern, daß der Prozessor mitten in einem Befehl umgeschaltet wird, ist Bit 0 zwischengespeichert, bis ein Taktzyklus abgelaufen ist.

Bei einem Z80–BIOS–Zugriff sind die beiden Memory–Status–Leitungen (MS0 und MS1) auf 0 und haben damit das System–ROM eingeschaltet. Für den Bereich von $001F – $0FFF sind die MS0– und MS1–Leitungen auf 1. Die beiden RAM–Banks können über die Adresse A16 eingeschaltet werden. A16 schaltet die beiden CAS0– und CAS1–Leitungen.

Im C64–Modus wird die Z80–Enable–Leitung immer auf 0 gehalten, unabhängig vom Zustand des Bits 0.

Die Bits 1 und 2 sind ungenutzt, aber für spätere Porterweiterungen gedacht. In der augenblicklichen Version sind beide Bits auf 1, wenn sie gelesen werden. Sie können aber nicht beschrieben werden.

Bit 3 ist das FSDIR–Kontroll–Bit. Es wird als Ausgabebit für den schnellen seriellen Disketten–Daten–Bus–Puffer und als Eingabebit für das Disk–Enable–Signal benutzt. Der MMU–Anschluß FSDIR zeigt den Zustand dieses Bits an, das nach dem Einschalten oder nach einem Systemreset auf 0 gesetzt ist.

Bit 3 ist als bidirektionaler Port eingerichtet, ähnlich wie ein Bit im MOS 6529–Port. Der Wert, der diesem Bit zugeordnet wird, wird ausgegeben und

wird auch wieder gelesen, wenn nicht von außen das Port auf 0 gezogen wird. In diesem Fall wird dieses Bit als 0 gelesen. Wird die externe Quelle entfernt, zeigt das Bit seinen vorherigen Zustand an.

Die Bits 4 und 5 sind das GAME– und EXROMIN–Eingabebit. Sie zeigen, ob die Steckmodulleitungen GAME oder EXROMIN, die im C64–Modus benutzt werden, belegt sind.

Bei C128–Steckmoduln werden diese Sense–Leitungen nicht benutzt, wodurch anhand des Zustands dieser Leitungen im C128–Modus erkannt werden kann, ob es sich um C64–Steckmodule handelt und der C64–Modus eingeschaltet werden muß. Diese Bits sind ebenfalls als bidirektionale Bits ausgelegt.

Das GAME–Bit kontrolliert die Umschaltung des Farbspeichers, wenn es als Ausgabebit geschaltet ist. Das Bit 5 (EXROMIN) wird im Augenblick nicht als Ausgabe genutzt.

Bit 6 schaltet zwischen den beiden Operationsmodi um. Dieses Bit ist nach dem Einschalten oder nach einem Systemreset auf 0 voreingestellt, d.h., der C128–Modus ist eingeschaltet, alle MMU–Register sind aktiviert und alle C128–Steuer–Leitungen sind vorbereitet. Wird dieses Bit auf 1 gesetzt, so wird die MMU ab– und der C64–Modus eingeschaltet.

Zu beachten ist, daß die MS3–Leitung den Zustand des Bits 6 invertiert darstellt.

Ist das Bit 7 als Eingabebit deklariert, zeigt es den Zustand der 40/80–Display–Taste an. Ist Bit 7 = 1, ist die Taste offen, ist es 0, ist die Taste gedrückt.

Der Anzeige–Modus wird entsprechend der Softwareinterpretation dieses Bits ein– oder ausgeschaltet. Als Ausgabebit wird dieses Bit gegenwärtig nicht genutzt.

B.2.4 Das RAM–Konfigurations–Register

Das RAM–Konfigurations–Register setzt die Segmentparameter für den Prozessor und für den Blockzeiger des VIC–Bausteins. Dieses Register ist im I/O–Bereich bei $D506 angelegt (s. Bild B.4).

Die Bits 0 und 1 bestimmen die Größe des auf die einzelnen Speicherbänke aufteilbaren RAMs. Vorausgesetzt, daß ein gemeinsamer RAM–Bereich

deklariert wurde, werden die RAM–Bank–Bits des Konfigurations–Registers grundsätzlich überlesen, da die ausgewählte RAM–Bank für nicht–gemeinsame Bereiche genutzt wird (die ROM– und I/O–Bereich–Bits des Konfigurationsregisters (s. Kapitel B.2.1) bleiben jedoch wirksam). Wenn der Wert der Bits 0 und 1 Null ist, werden 1 kBytes gemeinsamer RAM eingeschaltet. Ist nur das Bit 0 = 1, werden 4 kBytes, ist nur das Bit 1 = 1, werden 8 kBytes und sind beide Bits auf 1, werden 16 kBytes gemeinsamer RAM–Bereich eingeschaltet. Diese Bits wirken sich im C64–Modus nicht aus. Nach dem Einschalten oder nach einem System–Reset sind beide Bits auf 0 voreingestellt.

Bits 2 und 3 bestimmen, ob und wo ein gemeinsamer Speicherbereich angelegt wird. Sind beide Bits auf 0, wird kein gemeinsamer Bereich eingerichtet. Ist das Bit 2 auf 1, ist der untere RAM–Bereich als gemeinsamer Bereich deklariert. Ist Bit 3 auf 1, ist der obere RAM–Bereich als gemeinsamer Bereich eingerichtet. Sind beide Bits gesetzt, ist ein Teil des gemeinsamen Bereiches im unteren und der andere Teil im oberen RAM–Bereich angelegt. Beim Einschalten oder nach einem System–Reset sind beide Bits auf 0 gesetzt, so daß kein gemeinsamer Bereich vereinbart ist.

Bei der vorliegenden MMU–Version wird die Auswahl des gemeinsamen Bereiches durch die CAS0– und CAS1–Signale bestimmt. Das CAS0– wird auf 0 und das CAS1–Signal auf 1 gehalten.

Bit 4 und 5 werden z.Z. nicht genutzt, sind aber als Blockzeiger–Bits gedacht. Damit wird es zukünftig möglich sein, bis zu 1 MByte RAM in 256–kBytes-Blöcken zu adressieren. Sind beide Bits 0, ist der 1. 256 K–Block aktiviert. ist nur das Bit 4 = 1, ist der 2. Block aktiviert usw. Auf diese Weise lassen sich 4 Blöcke 256 kBytes adressieren.

Bits 6 und 7 sind die RAM–Block–Zeiger für den VIC–Baustein. Bei der vorliegenden MMU–Version wird das Bit 7 ignoriert. Das Bit 6 wird dazu benutzt, das CAS0–Signal auf 0 zu legen, wenn Bit 6 = 0 ist bzw., wenn Bit 6 = 1 ist, das CAS1–Signal auf 0 zu legen. So wird entweder Bank 0 oder 1 angewählt.

Beim 2–MHz–Betrieb übernimmt der 80–Zeichen–Baustein die Bildschirm-Steuerfunktionen, wobei der VIC–Baustein abgeschaltet wird. Das Abschalten wird von dem VIC-Baustein selber verursacht, indem das AEC- (Address-Enable–Control) Signal konstant hochgehalten wird.

Zu beachten ist, daß ein VIC–Zyklus erkannt wird, wenn AEC nach 0 gezogen wird. Dieses kann von einem DMA (Direct Memory Access) herrühren, der

die ACE–Leitung ebenfalls nach 0 zieht. Jeder DMA veranlaßt die MMU, in die VIC–Konfiguration umzuschalten.

B.2.5 Die Speicherseiten–Zeiger

Die Seitenzeiger (s. Bild B.4) sind vier Register, die eine unabhängige Adressierung der beiden untersten Speicherseiten von je 256 Bytes erlauben. Dies ist besonders nützlich, denn sie helfen, die Festlegung der Zero Page und des Stapelspeichers auf den physikalischen Adreßbereich $0000–$01FF, wie sie normalerweise für die 6500–Prozessor–Familie gilt, zu beseitigen.

Für die Verschiebung der Zero Page liefert die MMU die beiden Zero Page–Zeiger–Register P0H und P0L (Page 0 Pointer High, Page 0 Pointer Low). Bei einem beliebigen Zero Page–Zugriff ($0000–$00FF) werden die Bits 0 bis 3 des P0H–Registers und die mit dem P0H–Registern korrespondieren Adressen TA16 bis TA19 gesetzt.

In der vorliegenden Version kontrolliert das Bit 0 die Erzeugung des CAS0– oder des CAS1–Signals abhängig davon, ob das Bit 0 = 0 oder 1 ist. Die Bits 1 bis 3 werden ignoriert.

Die restlichen Bits sind 0, sie überdecken die RAM–Bank–Bits. Die ROM-Block– und die I/O–Block–Bits bestimmen, welche physikalische Speicherseite als Zero Page bei allen Zero–Page–Zugriffen erscheint. Wird das P0H–Register beschrieben, wird es solange zwischengespeichert, bis auch das P0L–Register beschrieben wird.

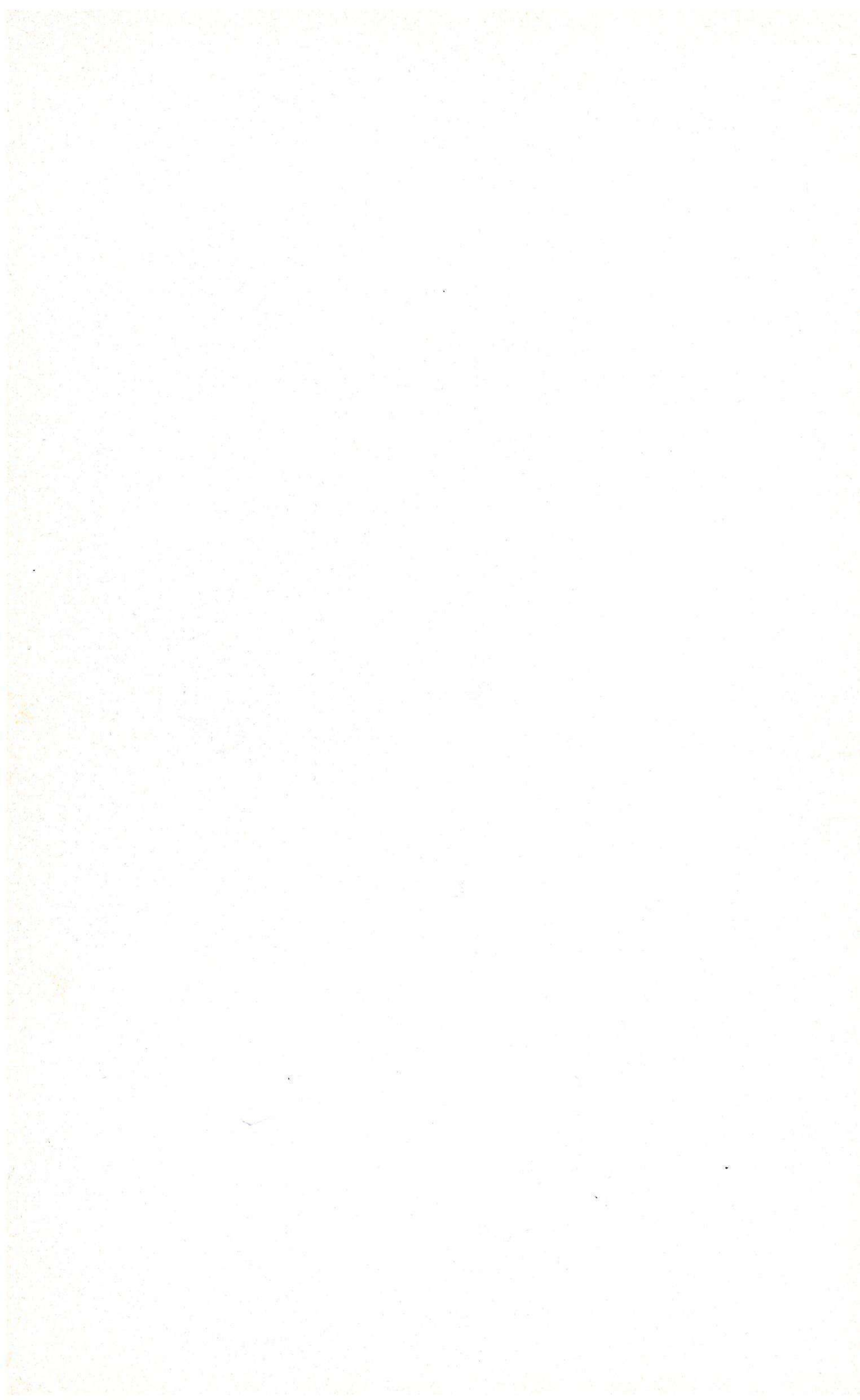
Die Bits 0–7 dieses Registers wirken auf die Adressen TA8–TA15, wenn auf eine deklarierte Zero Page zugegriffen wird. In diesem Fall wird automatisch auf die Original–Zero Page umgeschaltet. Ein Schreiben in dieses Register erzwingt einen Zero Page–Transfer, sobald der Taktimpuls auf 0 geht. Die beiden Register sind im I/O–Bereich angelegt, und zwar P0L bei $D507 und P0H bei $D508.

Die Register für die Verschiebung der Speicherseite 1 (Stapelspeicher) sind im I/O–Bereich für P1L bei $D509 und P1H bei $D50A angeordnet. Die Funktionen und die korrespodierenden Signale entsprechen denen von P0H und P0L. Die Registerpaare sind nach dem Einschalten oder nach einem System–Reset auf 0 voreingestellt, d.h. Zero Page und Stapelspeicher liegen physikalisch und logisch im Adreßbereich $01FF.

B.2.6 Das System–Versions–Register

Das letzte Register, das hier erwähnt werden soll, ist das Versions–Register (s. Bild B.4). Es ist im I/O–Bereich bei $D50B angelegt. Dieses Register ist ein Leseregister (read only).

Es enthält in dem niederwertigen Halbbyte eine Information über die aktuelle MMU–Version. In dem höherwertigen Halbbyte ist eine Information über den Speicherausbau enthalten. Damit ist es möglich, für zukünftige Versionen entwickelte Programme, die den Speicherausbau berücksichtigen, ablaufen zu lassen



Anhang C:

Maschinensprache-Monitor



Anhang C: Maschinensprache–Monitor

Für Anwender, die den Speicher des C128 direkt verwalten wollen oder mit Maschinensprache–Programmen arbeiten, enthält das Betriebssystem des C128 einen Monitor, durch den der Benutzer im C128–Modus jederzeit wichtige Informationen über den inneren Zustand des Computers erhalten kann.

Im allgemeinen handelt es sich dabei um Inhalte von Registern und Speicherstellen. Dabei werden zur besseren Übersicht alle Adressen und Speicherinhalte in hexadezimaler Schreibweise (hex) angegeben. Damit werden alle Registerinhalte als zweistellige und alle Adressen als vier– oder fünfstellige Hex–Zahlen angegeben. Die höchstwertige Hex–Ziffer einer fünfstelligen Adressangabe bezeichnet die Speicherkonfiguration, bei der der vorangestellte Monitorbefehl wirken soll. Es gibt folgende 16 Speicherkonfigurationen:

0	RAM–Bank 0
1	RAM–Bank 1
2	RAM–Bank 2 (zukünft. Erweiterung)
3	RAM–Bank 3 (zukünft. Erweiterung)
F	Betr.–System + BASIC, RAM 0, I/O, Zeichengenerator

Der Monitor wird durch den Befehl MONITOR des BASIC 7.0 im C128–Modus aufgerufen. Er wird ebenfalls aktiviert, wenn ein Maschinenspracheprogramm ausgeführt wird, das den Befehl BRK mit dem Befehlscode 0 enthält. Dabei wird folgende Bildschirmanzeige sichtbar:

```
MONITOR
    PC   SR AC XR YR SP
  FB000 00 00 00 00 F8
```

Die Abkürzungen in der oberen Reihe der Anzeige haben folgende Bedeutung:

PC: Der Programmzähler (program counter), der die Adresse des nächsten auszuführenden Befehls enthält.

SR: Der Inhalt des Statusregisters des Mikroprozessors 8502.

AC: Der Inhalt des Akkumulator–Registers des Mikroprozessors 8502.

XR: Der Inhalt des Indexregisters X des Mikroprozessors 8502.

YR: Der Inhalt des Indexregisters Y des Mikroprozessors 8502.

SP: Der Inhalt des Stapelzeigers (stack pointer) des Mikroprozessors 8502.

Der Maschinensprache–Monitor kennt eine ganze Reihe komfortabler Befehle, mit dem der Speicher des C128 bearbeitet werden kann, Speicherbereiche von/auf Kassette oder Diskette geladen/gespeichert und Maschinensprache–Programme assembliert, disassembliert, ausgeführt und getestet werden können. Nach einer Zusammenstellung aller möglichen Befehle wird jeder Befehl ausführlich und anhand von Beispielen beschrieben.

Befehl	Bedeutung
A (assemble)	Assembliert eine Zeile im 6502–Befehlscode.
C (compare)	Vergleicht zwei Speicherbereiche byteweise und zeigt Unterschiede an.
D (disassemble)	Disassembliert einen Speicherbereich mit Maschinensprache–Code.
F (fill)	Füllt einen Speicherbereich mit einem angegebenen Byte.
G (go)	Startet ein Maschinensprache–Programm bei einer angegebenen Adresse.
H (hunt)	Durchsucht einen Speicherbereich nach einer angegebenen Bytefolge.
L (load)	Lädt eine Programmdatei von Kassette oder Diskette.
M (memory)	Zeigt den Inhalt eines angegebenen Speicherbereiches hexadezimal an.
R (register)	Zeigt den Inhalt der Prozessorregister an.
S (save)	Speichert den angegebenen Speicherbereich auf Kassette oder Diskette.
T (transfer)	Verschiebt den Inhalt eines angegebenen Speicherbereiches.
V (verify)	Vergleicht einen Speicherbereich mit einer Programmdatei auf Kassette oder Diskette.
X (exit)	Beendet den Monitor und kehrt in den BASIC–Direktmodus zurück.
>	Modifiziert ein bis acht Speicherzellen ab der angegebenen Adresse.

Befehl	Bedeutung
.	Identisch mit dem A–Befehl.
;	Modifiziert die Prozessoregister.
@	Zeigt den Floppy–Disk–Status an oder überträgt einen Floppy–Disk–Befehl.

Für die Syntax der Beschreibungen der einzelnen Befehle gelten dieselben Regeln, die in Kapitel 4.5 für die BASIC–Befehle zusammengestellt wurden.

Adressen können grundsätzlich in jedem Zahlensystem angegeben werden (s.u.). Sind sie hexadezimal 1– bis 4–stellig, so beziehen sie sich auf die Speicherbank 0, sind sie hexadezimal 5–stellig, so gibt die höchstwertige Hexadezimalziffer die gewünschte Speicherbank an.

Der Monitor zeigt Adressen grundsätzlich 5–stellig hexadezimal an.

Je nach eingestellter Bildschirmbreite werden beim M–Befehl 8 bzw. 16 Byteinhalte je Bildschirmzeile sowohl hexadezimal als auch als Zeichen in ASCII–Codierung dargestellt.

Numerische Werte können beim Monitor durch Voranstellen von speziellen Symbolen in jedem Zahlensystem definiert werden:

kein Symbol	hexadezimale Werte (Voreinstellung)
$	hexadezimale Werte
+	dezimale Werte
&	oktale Werte
%	binäre Werte

Beispiel:

```
M +14500 +14550
```

Der Speicherbereich zwischen 14500 und 14550 dezimal in der Bank 0 wird angezeigt.

A Assemblieren

Format: **A *Adresse Befehl* [Operand]**

Zweck: Assembliert den angebenen Maschinensprache–Befehl bei der angegebenen Adresse.

Adresse Hexadezimale Speicheradresse, bei der der Befehl assembliert werden soll.

Befehl Mnemonische Schreibweise des Maschinensprache–Befehls (s. Programmierhandbuch für den Mikroprozessor 6502).

Operand Definiert, falls erforderlich, den Adressierungsmodus und muß hexadezimal angegeben werden.

Bemerkungen: Drücken der RETURN–Taste beendet die Eingabe der Befehlszeile und startet die Assemblierung. Die assemblierte Zeile überschreibt dabei die eingegebene.

Wird die RETURN–Taste ohne weitere Eingabe gedrückt, so wird der Assemblermodus beendet.

Es kann nur in RAM–Bereiche assembliert werden.

Werden Fehler erkannt, so wird ein Fragezeichen (?) angezeigt und der Cursor wird auf den Anfang der nächsten Zeile gesetzt. Mit Hilfe des Bildschirmeditors kann die Zeile korrigiert werden.

Beispiele:

```
A 1200 LDX #$0
A 1200 A2 00        LDX #$00
A 1202
```

C Speichervergleich

Format: **C *Adresse1 Adresse2 Adresse3***

Zweck: Vergleicht zwei Speicherbereiche und zeigt die Adressen der unterschiedlichen Speicherbytes an.

Adresse1 Hexadezimale Speicheradresse, die den Anfang des Bereiches definiert, gegen den verglichen wird.

Adresse2 Hexadezimale Speicheradresse, die das Ende des Bereiches definiert, gegen den verglichen wird.

Adresse3 Hexadezimale Speicheradresse, die den Anfang des Bereiches definiert, mit dem verglichen wird.

Bemerkungen: Für alle gefundenen Unterschiede werden die Adresse je nach Bildschirmbreite in 5 oder 10 Spalten angezeigt.

D Disassemblieren

Format: **D** [***Adresse1***] [**Adresse2**]

Zweck: Disassembliert einen Bereich mit Maschinensprachecode und zeigt dafür die mnemonischen Befehle zusammen mit dem Code an.

Adresse1 Hexadezimale Speicheradresse, die den Anfang des Bereiches definiert, der disassembliert werden soll.

Adresse2 Hexadezimale Speicheradresse, die das Ende des Bereiches definiert, der disassembliert werden soll.

Bemerkungen: Das Diasassemblierungs–Anzeigeformat unterscheidet sich von dem der Assemblierung darin, daß am Zeilenanfang ein Punkt (.) statt eines A ausgegeben wird. Außerdem wird der Maschinensprache–Code zusätzlich vor dem mnemonischen Befehl angezeigt.

Nicht identifizierbare Codes werden in der Befehlsspalte als ??? dargestellt.

Die Disassemblierungsliste kann jederzeit mit dem Bildschirmeditor modifiziert werden, wordurch automatisch der Assembler aktiviert wird (.–Befehl).

Werden die Adressangaben weggelassen, so werden 19 Bildschirmzeilen angezeigt.

Beispiel:

```
.D F6000 F6006
. F6000 A5 63       LDA $63
. F6002 D0 F3       BNE $5FF7
. F6004 A9 05       LDA #$05
. F6006 20 FB 4F    JSR $4FFB
```

F Speicherbereich füllen

Format: **F *Adresse1 Adresse2 Byte***

Zweck: Füllt einen Speicherbereich mit einem angegeben Byte.

Adresse1 Hexadezimale Speicheradresse, die den Anfang des Bereiches definiert, der gefüllt werden soll.

Adresse2 Hexadezimale Speicheradresse, die das Ende des Bereiches definiert, der gefüllt werden soll.

Byte Eine ein– oder zweistellige Hexadezimalzahl, die den Code angibt, mit dem der Bereich gefüllt werden soll.

Bemerkungen: Dieser Befehl ist zur Erzeugung bestimmter Datenstrukturen oder RAM–Bereiche hilfreich.

Beispiel:

```
F 0400 0518 EA
```

Der Bereich zwischen $0400 und $0518 in der Speicherbank 0 wird mit $EA (Code für den NOP–Maschinensprachebefehl) gefüllt.

G Programm starten

Format: **G** [***Adresse***]

Zweck: Startet ein Maschinenspracheprogramm bei einer angegebenen Adresse.

Adresse Hexadezimale Speicheradresse, bei der das Programm gestartet werden soll. Wird ***Adresse*** weggelassen, so beginnt die Programmausführung beim aktuellen Stand des Programmzählers, der mit dem R–Befehl (s. dort) angezeigt werden kann.

Bemerkungen: Ehe das Programm gestartet wird, werden die aktuellen Registerinhalte (durch den R–Befehl anzeigbar) gesichert.

Um nach der Ausführung des Maschinenspracheprogramms wieder den Monitor aufzurufen, muß der letzte Programmbefehl ein BRK–Befehl (Code $00) sein.

Beispiel:

```
G 140C
```

Das bei der Adresse $104C in der Speicherbank 0 beginnende Maschinenspracheprogramm wird gestartet.

H Speicherbereich durchsuchen

Format: **H *Adresse1 Adresse2 Daten***

Zweck: Durchsucht einen Speicherbereich nach einer angegebenen Bytefolge.

Adresse1 Hexadezimale Speicheradresse, die den Anfang des Bereiches definiert, der durchsucht werden soll.

Adresse2 Speicheradresse, die das Ende des Bereiches definiert, der durchsucht werden soll.

Daten Eine oder mehrere zweistellige Hexadezimalzahlen, getrennt durch Leerzeichen oder eine Zeichenkette, der ein Apostroph (') vorangestellt ist.

Bemerkungen: Es werden die Adressen aller Stellen des angegebenen Bereiches, bei denen die definierten Daten gefunden wurden, je nach eingestellter Bildschirmbreite in 5 bzw. 10 Spalten angezeigt.

Beispiel:

```
H A000 A100 4C FF A9
```

Durchsucht den Bereich zwischen $A000 und $A100 in der Speicherbank 0 nach der Bytefolge $4CFFA9.

```
H F4000 FFFFF 'MON
```

Durchsucht den Bereich zwischen $F4000 und FFFFF nach der Zeichenkette MON.

L Programmdatei laden

Format: L ***Dateiname*** [,***Geräteadr***] [,***Ladeadr***]

Zweck: Lädt eine Programmdatei von Kassette oder Diskette in den Speicher.

Dateiname Eine Zeichenkette von maximal 16 Zeichen Länge, eingeschlossen in Anführungszeichen ("), die den Dateinamen angibt.

Geräteadr Die Geräteadresse des externen Datenträgers, von dem geladen werden soll. Für Kassette muß 01 und für Floppy–Disk je nach dort eingestellter Adresse ein Wert zwischen 04 und 0F angegeben werden.

Ladeadr Eine gültige Speicheradresse, ab der geladen werden soll. Wird ***Ladeadr*** weggelassen, so wird die Datei ab der Adresse in der aktuellen Speicherbank geladen, ab der sie auch gespeichert wurde.

Bemerkungen: Es wird immer die gesamte Datei geladen.

Beispiele:

```
L"PROGR 1",08
```

Lädt die Programmdatei PROGR 1 von Diskette ab der Speicheradresse, ab der die Datei auch gespeichert wurde.

```
L"PROGRAMM",0A,2000
```

Lädt die Programmdatei PROGRAMM von Diskette in Floppy–Disk Nr. 10 ab Adresse $2000 in der aktuellen Speicherbank.

M Speicherinhalt anzeigen

Format: **M *Adresse1* [*Adresse2*]**

Zweck: Zeigt den Inhalt eines wählbaren Speicherbereiches in einer wählbaren Speicherbank auf dem Bildschirm an.

Adresse1 Hexadezimale Speicheradresse, die den Anfang des Bereiches definiert, der angezeigt werden soll. Wird ***Adresse1*** weggelassen, so wird der Speicherbereich ab dem aktuellen Programmzähler in der Speicherbank 0 in 12 Zeilen auf dem Bildschirm angezeigt

Adresse2 Hexadezimale Speicheradresse, die das Ende des Bereiches definiert, der angezeigt werden soll. Wird ***Adresse2*** weggelassen, so wird der Speicherbereich ab ***Adresse1*** in 12 Zeilen auf dem Bildschirm angezeigt.

Bemerkungen: Der Speicherbereich wird in folgendem Format angezeigt (Beispiel):

```
>1A048 41 E7 00 AA AA 00 98 56    :A!.**. .V
```

Nach dem >–Zeichen folgt die vollständige Adresse des ersten Bytes der angezeigten Zeile, dann je nach eingestellter Bildschirmbreite die Inhalte von 8 bzw. 16 Speicherbytes in hexadezimaler Codierung und dann die ASCII-Äquivalente dieser 8 bzw. 16 Bytes in inverser Darstellung. Nicht darstellbare Codes werden hier durch Punkte (.) angezeigt.

Der angezeigte Speicherinhalt kann mit Hilfe des Bildschirmeditors verändert werden. Dazu wird der Cursor auf die betreffende Byteanzeige im hexadezimalen Feld gestellt und die Änderung hexadezimal eingegeben. Durch Drücken der RETURN–Taste wird immer eine Bildschirmzeile übernommen (>–Befehl).

Eingabefehler oder der Versuch, ROM–Bereiche zu modifizieren führen zur Anzeige eines Fragezeichens (?) an der aktuellen Cursor–Position.

Es werden immer vollständige Bildschirmzeilen angezeigt.

Beispiel:

```
.M F4540 F4547
> F4540 54 52 4F CE 54 52 4F 46: TRONTROF
```

R Prozessorregister anzeigen und ändern

Format: **R**

Zweck: Zeigt den aktuellen Inhalt sämtlicher Register des Mikroprozessors 8502 (Programmzähler PC, Statusregister SR, Akkumulator AC, Indexregister XR und YR, Stapelzeiger SP) sowie den Interrupt–Vektor IRQ an.

Bemerkungen: Die Registerinhalte können geändert werden, indem die Einträge der Registeranzeige mit dem Bildschirmedditor geändert werden. Drücken der RETURN–Taste übernimmt diese Einträge (;–Befehl).

Beispiel:

```
.R
   PC      SR AC XR YR SP
.; FB000   00 00 00 00 F8
```

PC SR AC XR YR SP
E0076 7A 56 A1 50 F5

S Speicherbereich sichern

Format: **S "[*Laufw:*]*Name*", *Geräteadr*, *Adresse1*, *Adresse2***

Zweck: Sichert einen angegebenen Speicherbereich auf Kassette oder Diskette als Programmdatei.

Laufwerk Wird auf Diskette gespeichert, so muß hier bei Doppellaufwerken unbedingt 0 oder 1 eingegeben werden. Bei Einzellaufwerken kann die Laufwerksangabe entfallen. Soll auf Kassette gespeichert werden, muß die Angabe ***Laufw:*** entfallen.

Name Eine Zeichenkette von maximal 16 Zeichen bei Kassetten– bzw. 14 Zeichen bei Disketten–Sicherung, die den Dateinamen angibt.

Geräteadr Die Geräteadresse des externen Datenspeichers, auf dem der Speicherbereich gesichert werden soll. Für Kassette muß 01 und für Floppy–Disk je nach dort eingestellter Adresse ein Wert zwischen 04 bis 0F angegeben werden.

Adresse1 Hexadezimale Speicheradresse , die den Anfang des Bereiches definiert, der gesichert werden soll.

Adresse1 Hexadezimale Speicheradresse + 1, die das Ende des Bereiches definiert, der gesichert werden soll.

Bemerkungen: Die ersten beiden Bytes der gesicherten Datei enthalten die Startadresse des gesicherten Speicherbereiches in der Anordnung höherwertiges Byte/niederwertiges Byte.

Beispiele:

```
S"SPIEL",01,1B000,1B800
```

Speichert den Adressbereich von $B000 bis $B7FF einschließlich in Speicherbank 1 unter dem Namen SPIEL als Programmdatei auf Kassette.

```
S"0:MEM",0A,2000,2500
```

Speichert den Bereich $2000 bis $24FF der Speicherbank 0 auf Diskette in Laufwerk 0 der Floppy–Disk mit der Geräteadresse 10.

T Speicherbereich übertragen

Format: **T *Adresse1 Adresse2 Adresse3***

Zweck: Überträgt den Inhalt eines angegebenen Speicherbereiches in einen anderen Speicherbereich.

Adresse1 Hexadezimale Speicheradresse, die den Anfang des Bereiches definiert, dessen Inhalt verschoben werden soll.

Adresse2 Hexadezimale Speicheradresse, die das Ende des Bereiches definiert, dessen Inhalt verschoben werden soll.

Adresse3 Hexadezimale Speicheradresse, die den Anfang des Bereiches definiert, in den der Inhalt übertragen werden soll.

Bemerkungen: Speicherinhalte können von höheren zu niederen Adressen und umgekehrt verschoben werden und es können die Inhalte beliebig großer Speicherbereiche verschoben werden.

Während der Übertragung wird eine automatische Prüfung der übertragenen Bytes durchgeführt, um Speicherfehler anzeigen zu können.

Beispiel:

```
T 1400 1600 1401
```

Verschiebt den Inhalt des Bereiches $1400 bis $1600 in der Speicherbank 0 um ein Byte nach “oben“.

V Programmdatei verifizieren

Format: **V *Dateiname*** [**,*Geräteadr***] [**,*Anfadr***]

Zweck: Vergleicht eine Programmdatei auf Kassette oder Diskette mit einem Speicherbereich.

Dateiname Eine Zeichenkette von maximal 16 Zeichen Länge, eingeschlossen in Anführungszeichen ("), die den Dateinamen angibt.

Geräteadr Die Geräteadresse des externen Datenträgers, von dem verglichen werden soll. Für Kassette muß 01 und für Floppy–Disk je nach dort eingestellter Adresse ein Werrt zwischen 04 bis 0F angegeben werden.

Anfadr Eine gültige Speicheradresse, ab der verglichen werden soll. Wird ***Anfadr*** weggelassen, so wird der Speicherbereich ab der Adresse mit der Datei verglichen ,ab der diese auch gespeichert wurde.

Bemerkungen: Es wird immer die gesamte Datei verglichen.

Nach dem Drücken der RETURN–Taste wird bei diesem Befehl die Meldung VERIFYING angezeigt. Wird fehlende Übereinstimmung bei einem Byte gefunden, so wird zusätzlich die Meldung ERROR angezeigt.

Beispiele:

```
.V "PROGR 1",08
```

Vergleicht die Programmdatei PROGR 1 mit dem Speicherbereich ab der Adresse in der aktuellen Speicherbank, ab der die Datei auch gespeichert wurde.

X Rückkehr in den BASIC–Direktmodus

Format: **X**

Zweck: Beendet den Monitorbetrieb und verzweigt zurück in den BASIC–Direktmodus.

Beispiel:

.X
READY.

> Speicherinhalte definieren

Format: **> *Adresse Byte1*** [***Byte2 . . . Byten***]

Zweck: Definiert je nach eingestellter Bildschirmbreite die Inhalte von bis zu 8 bzw. 16 aufeinanderfolgenden Speicherzellen ab einer angegebenen Adresse.

Adresse Hexadezimale Speicheradresse, ab der Speicherzelleninhalte definiert werden sollen.

Byte1 . . . Byten Bis zu 8 bzw. 16 Bytes in 2–stelliger hexadezimaler Schreibweise getrennt durch eine Leerstelle, die im Speicher abgelegt werden sollen.

Bemerkungen: Unabhängig von der Anzahl der eingegebenen Bytes wird nach Ausführung dieses Befehls immer die ganze Bildschirmzeile von 8 bzw. 16 Speicherbyteinhalten angezeigt.

Dieser Befehl wird bei der Modifizierung der Bildschirmanzeige des M–Befehls (s. dort) automatisch ausgeführt.

Beispiel:

```
> 02200 41 42 43
```

Ab Adresse 2200 in der Speicherbank 0 werden die Bytes $41, $42 und $43 abgelegt.

. Assemblieren

Dieser Befehl hat dieselbe Wirkung wie der A–Befehl (s. dort).

@ Floppy–Disk–Bedienung

Format: @ [*Geräteadr*][,[*Floppybefehl*]]

Zweck: Zeigt den Status der angeschlossenen Floppy–Disk an (nur @ [*Geräteadr*]) oder überträgt eine angegebene Bedienungszeichenkette an die Floppy–Disk.

Geräteadr Ein Wert zwischen 08 und 0F für die Geräteadresse der angeschlossenen Floppy–Disk. Voreingestellt ist hier 08.

Floppybefehl Eine gültige Floppy–Disk–Befehlszeichenkette (s.a. Kapitel 6 und Floppy–Disk–Handbuch).

Beispiele:

```
@
00, OK,00,00
```

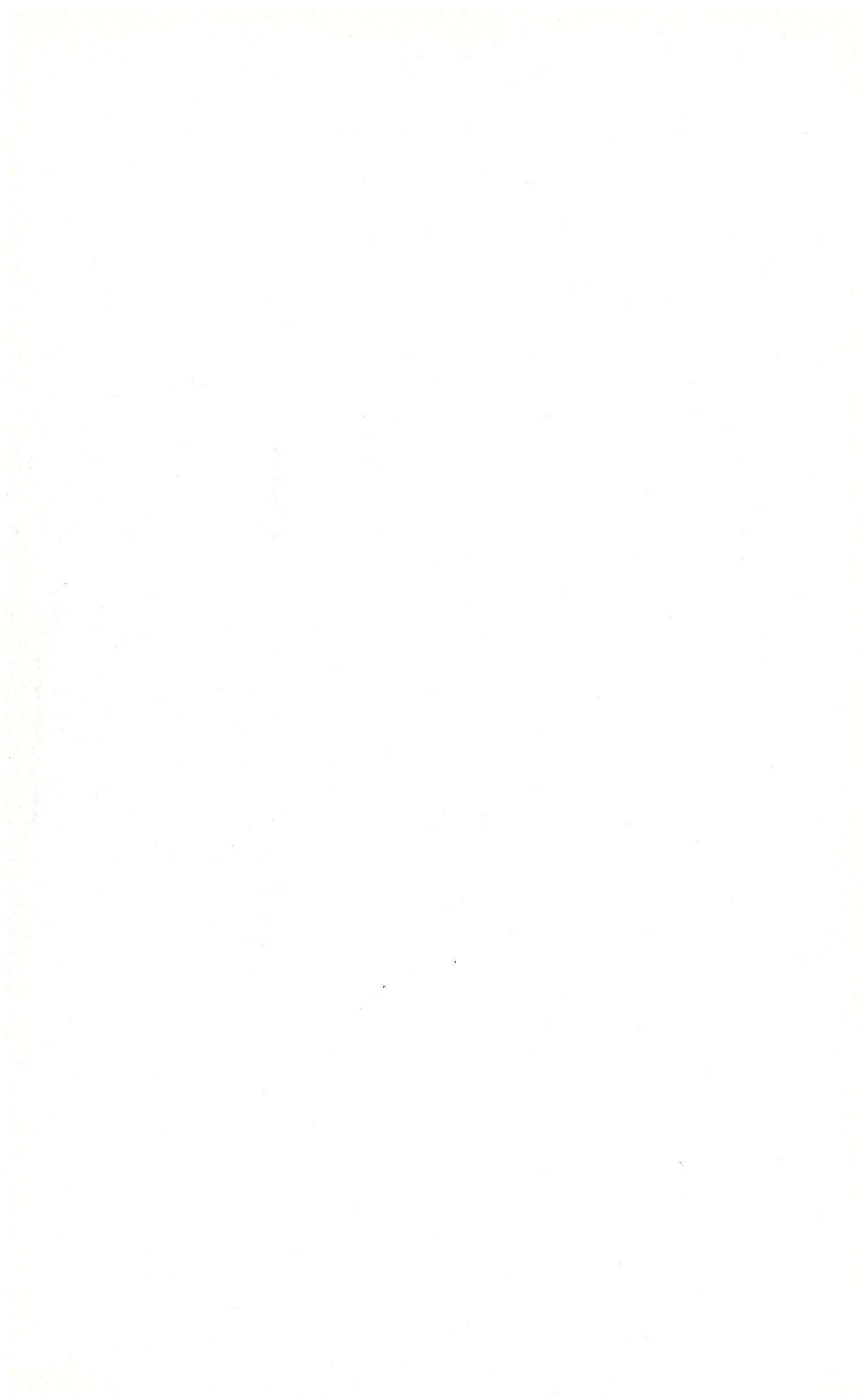
Der Floppy–Disk–Status wurde abgefragt.

```
@,S0:PRG*
01, FILES SCRATCHED,02,00
```

Auf der Diskette in Laufwerk 0 der Floppy–Disk mit der Geräteadresse 8 wurden zwei Dateien, deren Namen mit PRG beginnen, gelöscht.

```
@8,$
```

Das Inhaltsverzeichnis der Diskette in der Floppy–Disk mit der Geräteadresse 8 wird angezeigt.

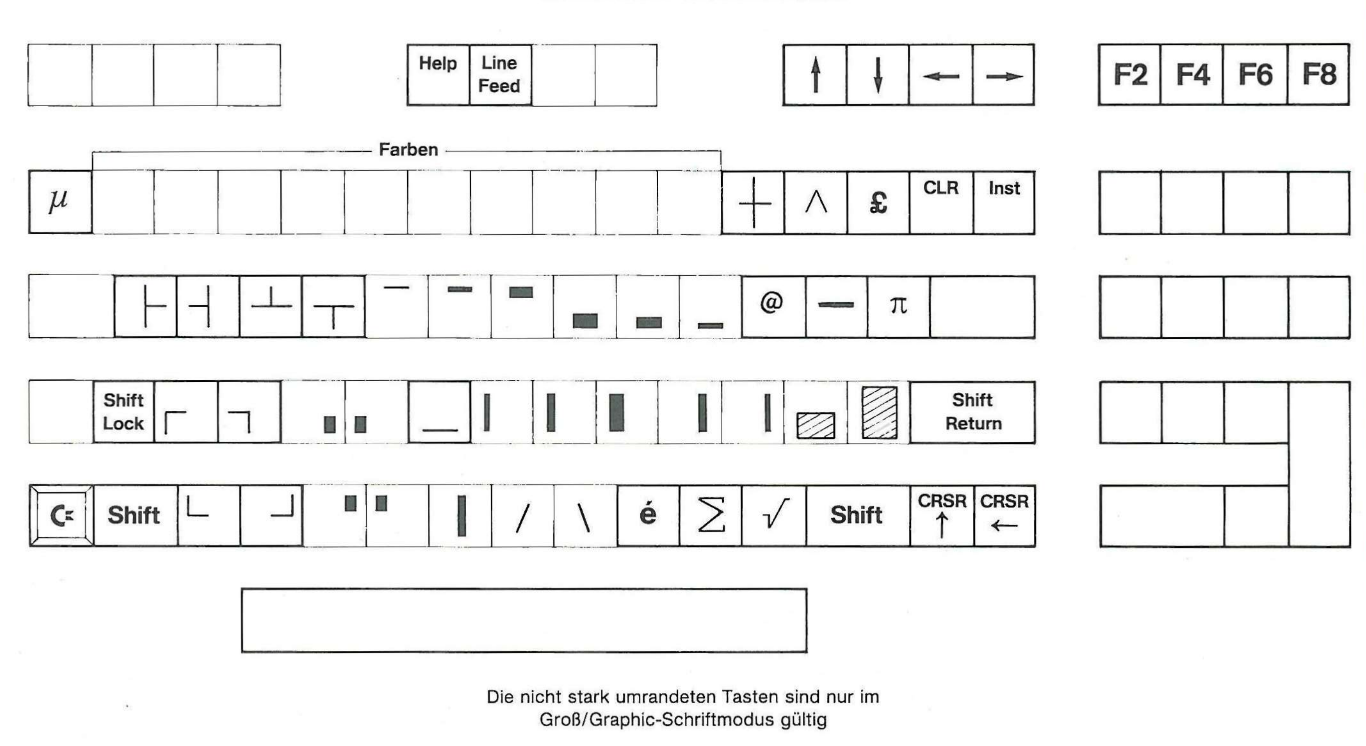


Anhang D:

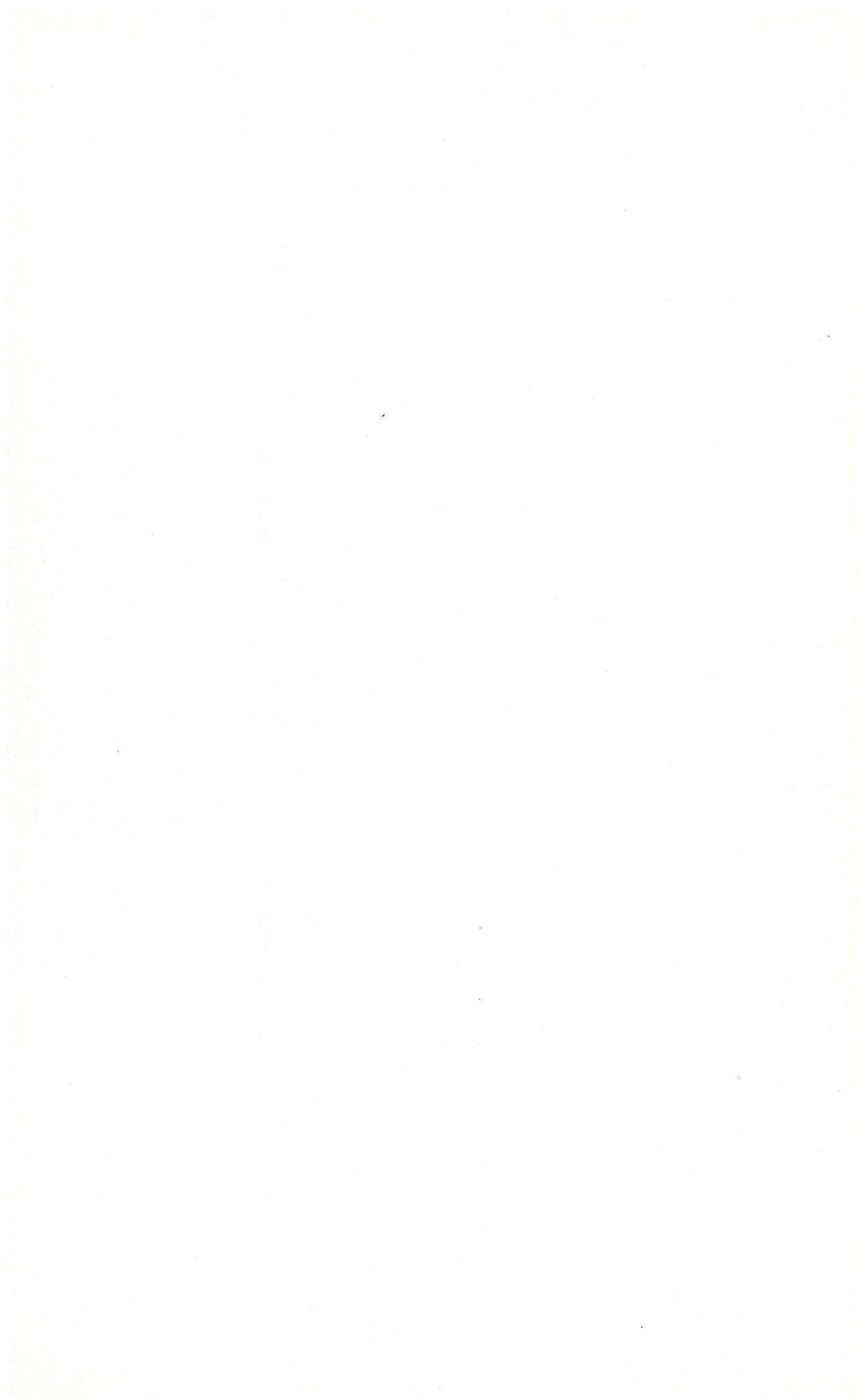
Besonderheiten der DIN-Tastatur



Tastaturbelegung bei gleichzeitigem Drücken der Commodore-Taste im DIN Modus



Die nicht stark umrandeten Tasten sind nur im Groß/Graphic-Schriftmodus gültig



Anhang E:

Registerzuordnungen beim SID und VIC

KONTROLLREGISTER FÜR TONERZEUGUNG

Basisadresse des SID: $54272_{Dez} = D400_{Hex}$

REGISTER STIMME 1	2	3	INHALT				
0	7	14	FREQUENZ, LO-BYTE (0 ... 255)				
1	8	15	FREQUENZ, HI-BYTE (0 ... 255)				
2	9	16	TASTVERHÄLTNIS, LO-BYTE (0 ... 255)	(NUR FÜR RECHTECK)			
3	10	17	TASTVERHÄLTNIS, HI-BYTE (0 ... 15)				
4	11	18	Wellenform:	RAUSCHEN 129	RECHTECK 65	SÄGEZAHN 33	DREIECK 17
5	12	19	ANSCHLAG 0*16 (hart) ... 15*16 (weich)	ABSCHWELLEN 0 (hart) ... 15*16 (weich)			
6	13	20	HALTEN 0*16 (stumm) ... 15*16 (laut)	AUSKLINGEN 0 (schnell) ... 15 (langsam)			
24	24	24	LAUTSTÄRKE: 0 (stumm) ... 15 (volle Lautstärke)				

Weitere Register des SID

REGISTER	INHALT							
21	GRENZFREQUENZ FILTER, LO-BYTE (0 ... 7)							
22	GRENZFREQUENZ FILTER, HI-BYTE (0 ... 255)							
23	RESONANZ 0 (keine) ... 15*16 (stark)				FILTER EINSCHALTEN: extern 8	Sti 3 4	Sti 2 2	Sti 1 1
24	FILTER-MODUS: Sti 3 aus 128	Hoch Paß 64	Band Paß 32	Tief Paß 16	LAUTSTÄRKE 0 (stumm) ... 15 (laut)			

Mit Hilfe dieser Einstellungen können Sie verschiedene Instrumente nachahmen:

Instrument	Wellenform		Anschlag	Halten	Tastverhältnis
Piano	Puls	65	9	0	HI-0, LO-255
Flöte	Dreieck	17	96	0	
Cembalo	Sägezahn	33	9	0	
Xylophon	Dreieck	17	9	0	
Orgel	Dreieck	17	0	240	
Akkordeon	Dreieck	17	102	0	
Trompete	Sägezahn	33	96	0	

BEMERKUNG: Die Einstellungen für die Hüllkurve sollten immer gePOKEt werden **bevor** die Wellenform gePOKEt wird.

SPRITE REGISTER ZUORDNUNG

Basisadresse VIC = 53248_{Dez} = $D000_{Hex}$

Register # Dez	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	S0X7	S0X6	S0X5	S0X4	S0X3	S0X2	S0X1	S0X0	SPRITE 0 X
1	1	S0Y7							S0Y0	SPRITE 0 Y
2	2	S1X7							S1X0	SPRITE 1 X
3	3	S1Y7							S1Y0	SPRITE 1 Y
4	4	S2X7							S2X0	SPRITE 2 X
5	5	S2Y7							S2Y0	SPRITE 2 Y
6	6	S3X7							S3X0	SPRITE 3 X
7	7	S3Y7							S3Y0	SPRITE 3 Y
8	8	S4X7							S4X0	SPRITE 4 X
9	9	S4Y7							S4Y0	SPRITE 4 Y
10	A	S5X7							S5X0	SPRITE 5 X
11	B	S5Y7							S5Y0	SPRITE 5 Y
12	C	S6X7							S6X0	SPRITE 6 X
13	D	S6Y7							S6Y0	SPRITE 6 Y
14	E	S7X7							S7X0	SPRITE 7 X
15	F	S7Y7							S7Y0	SPRITE 7 Y
16	10	S7X8	S6X8	S5X8	S4X8	S3X8	S2X8	S1X8	S0X8	Höchste Bits der X-Werte
17	11	RC8	EC5	BSM	BLNK	RSEL	YSCL2	YSCL1	YSCL0	
18	12	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	RASTER
19	13	LPX7							LPX0	LIGHT PEN X
20	14	LPY7							LPY0	LIGHT PEN Y
21	15	SE7							SE0	SPRITE ENABLE (EIN/AUS)
22	16	N.C.	N.C.	RST	MCM	CSEL	XSCL2	XSCL1	XSCL0	
23	17	SEXY7							SEXY0	SPRITE EXPAND Y

Register # Dez	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
24	18	VS13	VS12	VS11	CB13	CB12	CB11	CB10	N.C.	Speicherbereich für Bildschirmzeichen
25	19	IRQ	N.C.	N.C.	N.C.	LPIRQ	ISSC	ISBC	RIRQ	Interupt Request's
26	1A	N.C.	N.C.	N.C.	N.C.	MLPI	MISSC	MISBC	MRIRQ	Interupt Request MASKS
27	1B	BSP7							BSP0	Hintergrund Sprite PRIORITÄT
28	1C	SCM7							SCM0	MULTICOLOR SPRITE SELECT
29	1D	SEXX7							SEXX0	SPRITE EXPAND X
30	1E	SSC7							SSC0	Sprite-Sprite KOLLISION
31	1F	SBC7							SBC0	Sprite-Hintergrund KOLLISION
		FARBINFORMATION								
32	20									Bildrand
33	21									Hintergrund 0
34	22									Hintergrund 1
35	23									Hintergrund 2
36	24									Hintergrund 3
37	25					Sprite Multicolor {				SMC0
38	26									SMC1
39	27									Farbe Sprite 0
40	28									Farbe Sprite 1
41	29									Farbe Sprite 2
42	2A									Farbe Sprite 3
43	2B									Farbe Sprite 4
44	26									Farbe Sprite 5
45	2D									Farbe Sprite 6
46	2E									Farbe Sprite 7

Die Farbinformation entnehmen Sie bitte der Tabelle auf Seite 139. Im Multicolor-Modus dürfen nur die Farbcodes 0 . . . 7 benutzt werden.

80-Zeichen Video-Controller 8563

Der Video-Controller 8563 kann über die Adressen $D600 und $D601 angesprochen werden. Die Übergabe eines Wertes an den Controller (Nur C 128 Betriebsart) geschieht folgendermaßen:

Übergabe der Registernummer (0-36) an Adresse $D600 (54784 dez.) und Übergabe des Wertes an Adresse $D601 (54785 dez.)

Bsp.: Ändern des Vordergrund-Farbregisters 26

BANK15:POKE DEC("D600"),26:POKE DEC("D601"),9

Reg. (dez.)	Bits	Funktion
0	76543210	Gesamte Anzahl der Zeichen zwischen aufeinander-folgenden horizontalen Synchronimpulsen minus 1
1	76543210	Anzahl der dargestellten Zeichen pro Zeile
2	76543210	Position des horizontalen Synchronimpulses
3	7654	Breite des vertikalen Synchronimpulses (Anzahl der Bildschirmzeilen)
	3210	Breite des horizontalen Synchronimpulses (Zeichengenerator-Taktzyklen plus 1)
4	76543210	Gesamtzahl Zeichenzeilen minus 1 (Siehe auch Register 5)
5	. . .43210	Feinabgleich (Anzahl der Bildzeilen)
6	76543210	Anzahl der dargestellten Zeilen
7	76543210	Position des vertikalen Synchronimpulses (Anzahl Zeichenzeilen)
8	10	Anzeigeform X0: Kein Zeilensprung 01: Zeilensprung 11: Zeilensprung doppelte Dichte
9	. . .43210	Anzahl der Bildschirmzeilen pro Zeichenzeile minus 1
10	.65	Cursor Modus: 00: Cursor blinkt nicht 01: Kein Cursor sichtbar 10: Cursor blinkt mit 1/16 der Bildwiederholfrequenz 11: Cursor blinkt mit 1/32 der Bildwiederholfrequenz
	. . .43210	Cursor Startadresse

80-Zeichen Video-Controller 8563

Reg. (dez.)	Bits	Funktion
11	. . .43210	Cursor Endadresse
12	76543210	Startadresse des Bildschirmspeichers (High Byte)
13	76543210	Startadresse des Bildschirmspeichers (Low Byte)
14	76543210	Cursor-Position im Bildschirmspeicher (High Byte)
15	76543210	Cursor Position im Bildschirmspeicher (Low Byte)
16	76543210	Vertikale Light-Pen Position (In Zeichen-Zeilen plus 1)
17	76543210	Horizontale Light-Pen Position (In Zeichenzeilen plus 8)
18	76543210	Bildschirmspeicher-Adresse auf die sich der nächste CPU- Lese-, oder Schreibbefehl bezieht (High Byte)
19	76543210	Siehe Register 18 (Low Byte)
20	76543210	Startadresse des Attribut-Speichers (High Byte)
21	76543210	Startadresse des Attribut-Speichers (Low Byte)
22	7654	Zeichenbreite in Pixel minus 1 (Inklusive Zeichenzwischenraum)
	3210	Zeichenbreite in Pixel minus 1 (Ohne Zeichenzwischenraum)
23	. . .43210	Anzahl der Bildschirmzeilen pro Zeichenzeile minus 1 ohne vertikalen Zeichenabstand
24	7	Block Copy 0: Block wird geschrieben nach einem Schreiben in Register 30 1: Block wird kopiert nach einem Schreiben in Register 30
	.6	0: Normaldarstellung 1: Reversdarstellung
	. .5	Blinkfrequenz der Buchstaben 0: Blinkfrequenz = 1/16 der Bildwiederholfrequenz 1: Blinkfrequenz = 1/32 der Bildwiederholfrequenz

80-Zeichen Video-Controller 8563

Reg. (dez.)	Bits	Funktion
	. . .43210	Anzahl der Bildschirmzeilen, die der Bildschirm nach oben geschoben wird. Der Inhalt dieses Registers sollte zwischen 0 und dem Inhalt des Registers 9 liegen
25	7	Betriebsart 0: Textdarstellung 1: Bit-Mapped-Darstellung
	.6	Attribute 0: Attribute abgeschaltet (Vordergrundfarbe durch Register 26 bestimmt) 1: Vordergrundfarbe und Attribut sind im Attribut-Byte enthalten
	. .5	Semigrafik Betriebsart 0: Der horizontale Zwischenraum zweier Zeichen wird in der Hintergrundfarbe dargestellt 1: Der horizontale Zwischenraum zweier Zeichen wird in der gleichen Farbe dargestellt, wie das letzte normal dargestellte Zeichen.
	. . .4 . . .	Pixelgröße 0: Pixelgröße = 1 DOT-Takt 1: Pixelgröße = 2 DOT-Takte Horizontale Parameter müssen angepaßt werden
	3210	Anzahl der Pixel, die der Bildschirm nach links geschoben wird. Der Wert dieses Registers sollte zwischen dem Wert von Register 22 (Bit 4-7) und 0 (Inklusive) sein. Register 22 (Bits 0-3) sollte um den Wert dieser 4 Bits erhöht werden, um eine konstante Zeichenbreite zu erreichen
26	7654	Vordergrundfarbe (RGBI), wenn Bit 6 von Register 25 = 0
	3210	Hintergrundfarbe für alle Zeichen und Rahmenfarbe
27	76543210	Wert, der zum Bildschirm- und Attributzeiger nach jeder Zeile aufaddiert wird um die nächste Zeile zu erhalten

80-Zeichen Video-Controller 8563

Reg. (dez.)	Bits	Funktion
28	765.....	Adresse des Zeichengenerators (Bits 13-15). Vorher sollte Register 9 gesetzt werden
	...4....	DRAM-Type (4416/4164) 0: 4416 1: 4164
29	...43210	Bildschirmzeilen-zähler für Unterstreichen
30	76543210	Anzahl der Block-Write oder Block-Copy Zyklen. Eine 0 entspricht 256 Zyklen. Das Verändern dieses Registers startet den Block-Zyklus
31	76543210	Daten für Block-Write und normales ‚Write'. Beim Lesen des Registers wird die durch Register 18/19 bestimmte Bildschirmposition übergeben
32	76543210	Block-Copy Quelladresse (Bits 15-8). Die Zieladresse wird in Register 18/19 erwartet
33	76543210	Block-Copy Quelladresse (Bits 7-0)
34	76543210	Anzahl der Zeichen vom Beginn der dargestellten Zeile bis zur positiven Flanke des Display-enable-pins
35	76543210	Anzahl der Zeichen vom Beginn der dargestellten Zeile bis zur negativen Flanke des Display-enable-pins
36	3210	Anzahl der Refresh-Zyklen pro Bildschirmzeile



Anhang F:

Musiknotentabelle



WERTE FÜR MUSIK-NOTEN

In diesem Anhang finden Sie eine vollständige Liste der Noten, zugehörigen Frequenzen und Frequenzparameter und der Werte, die in die Register FREQ HI und FREQ LO des Klangchips gePOKEt werden müssen, um den gewünschten Ton zu erzeugen.

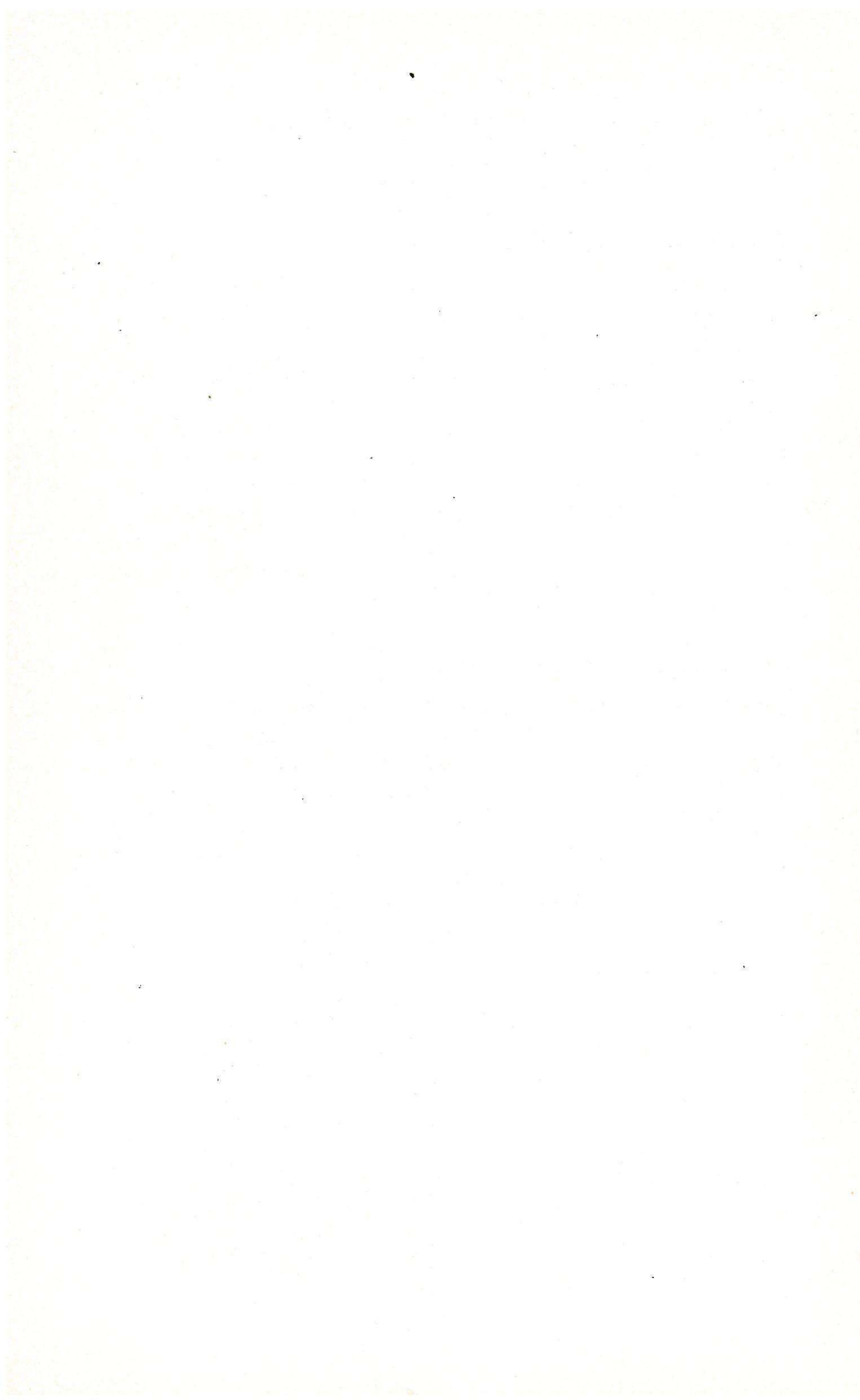
Nr.	Note-Oktave	Frequenz(Hz)	Parameter	Hi-Byte	Lo-Byte
0	C-0	16.4	278	1	22
1	C#-0	17.3	295	1	39
2	D-0	18.4	313	1	57
3	D#-0	19.4	331	1	75
4	E-0	20.6	351	1	95
5	F-0	21.8	372	1	116
6	F#-0	23.1	394	1	138
7	G-0	24.5	417	1	161
8	G#-0	26.0	442	1	186
9	A-0	27.5	468	1	212
10	A#-0	29.1	496	1	240
11	H-0	30.9	526	2	14
12	C-1	32.7	557	2	45
13	C#-1	34.6	590	2	78
14	D-1	36.7	625	2	113
15	D#-1	38.9	662	2	150
16	E-1	41.2	702	2	190
17	F-1	43.7	743	2	231
18	F#-1	46.2	788	3	20
19	G-1	49.0	834	3	66
20	G#-1	51.9	884	3	116
21	A-1	55.0	937	3	169
22	A#-1	58.3	992	3	224
23	H-1	61.7	1051	4	27
24	C-2	65.4	1114	4	90
25	C#-2	69.3	1180	4	156
26	D-2	73.4	1250	4	226
27	D#-2	77.8	1325	5	45
28	E-2	82.4	1403	5	123
29	F-2	87.3	1487	5	207
30	F#-2	92.5	1575	6	39
31	G-2	98.0	1669	6	133
32	G#-2	103.8	1768	6	232
33	A-2	110.0	1873	7	81
34	A#-2	116.5	1985	7	193
35	H-2	123.5	2103	8	55
36	C-3	130.8	2228	8	180
37	C#-3	138.6	2360	9	56
38	D-3	146.8	2500	9	196
39	D#-3	155.6	2649	10	89
40	E-3	164.8	2807	10	247
41	F-3	174.6	2974	11	158
42	F#-3	185.0	3150	12	78
43	G-3	196.0	3338	13	10
44	G#-3	207.7	3536	13	208
45	A-3	220.0	3746	14	162
46	A#-3	233.1	3969	15	129
47	H-3	246.9	4205	16	109

Nr.	Note-Oktave	Frequenz(Hz)	Parameter	Hi-Byte	Lo-Byte
48	C-4	261.6	4455	17	103
49	C#-4	277.2	4720	18	112
50	D-4	293.7	5001	19	137
51	D#-4	311.1	5298	20	178
52	E-4	329.6	5613	21	237
53	F-4	349.2	5947	23	59
54	F#-4	370.0	6301	24	157
55	G-4	392.0	6676	26	20
56	G#-4	415.3	7072	27	160
57	A-4	440.0	7493	29	69
58	A#-4	466.2	7939	31	3
59	H-4	493.9	8411	32	219
60	C-5	523.3	8911	34	207
61	C#-5	554.4	9441	36	225
62	D-5	587.3	10002	39	18
63	D#-5	622.3	10597	41	101
64	E-5	659.3	11227	43	219
65	F-5	698.5	11894	46	118
66	F#-5	740.0	12602	49	58
67	G-5	784.0	13351	52	39
68	G#-5	830.6	14145	55	65
69	A-5	880.0	14986	58	138
70	A#-5	932.3	15877	62	5
71	H-5	987.8	16821	65	181
72	C-6	1046.5	17821	69	157
73	C#-6	1108.7	18881	73	193
74	D-6	1174.7	20004	78	36
75	D#-6	1244.5	21193	82	201
76	E-6	1318.5	22454	87	182
77	F-6	1396.9	23789	92	237
78	F#-6	1480.0	25203	98	115
79	G-6	1568.0	26702	104	78
80	G#-6	1661.2	28290	110	130
81	A-6	1760.0	29972	117	20
82	A#-6	1864.7	31754	124	10
83	H-6	1975.5	33642	131	106
84	C-7	2093.0	35643	139	59
85	C#-7	2217.5	37762	147	130
86	D-7	2349.3	40008	156	72
87	D#-7	2489.0	42387	165	147
88	E-7	2637.0	44907	175	107
89	F-7	2793.8	47578	185	218
90	F#-7	2960.0	50407	196	231
91	G-7	3136.0	53404	208	156
92	G#-7	3322.4	56580	221	4
93	A-7	3520.0	59944	234	40
94	A#-7	3729.3	63508	248	20

Sie sind nicht an die Werte dieser Tabelle gebunden! Wenn Sie mehrere Stimmen benutzen, sollten Sie sogar bewußt die zweite und dritte Stimme etwas „verstimmen“, d. h. das Lo-Byte aus der Tabelle leicht (!) abändern. Sie bekommen so einen volleren Klang.

Anhang G:

Besonderheiten im C64-Modus
(Funkt.-Tasten, Grafik)



ABFRAGE DER DREHREGLER UND STEUERKNÜPPEL

Abfrage der Drehregler (Paddles):

Sind die Drehkegler in Port 1 eingesteckt, so werden die Werte, die den Stellungen der Potentiometer entsprechen, in die Adressen 54297 und 54298 eingelesen.

Das Drücken der Feuerknöpfe wird in der Adresse 56321 registriert. Dort wird Bit 2 bzw. Bit 3 gelöscht.

Die Drehregler in Port 2 benutzen für die Abfrage der Feuerknöpfe die Adresse 56320. Die Potentiometerpositionen werden in denselben Adressen registriert wie bei Port 1.

Die Umschaltung auf Port 2 geschieht durch Setzen von Bit 7 in Adresse 56320. Da dieses Bit jedoch bei jedem Interrupt zurückgesetzt wird, ist die Abfrage der Paddles am Port 2 nur in Verbindung mit einem Maschinenprogramm möglich.

Abfrage der Steuerknüppel (Joysticks)

Die Abfrage der Joysticks geschieht in den Adressen 56321 (Port 1) und 56320 (Port 2). Den 4 Richtungen des Joysticks (oben, unten, links, rechts) entspricht das Löschen der Bits Ø, 1, 2 und 3. Bei Betätigen der Feuerknöpfe wird Bit 4 im jeweiligen Register gelöscht.

BELEGUNG DER FUNKTIONSTASTEN

Die Funktionstasten (f1 bis f8) entsprechen den ASCII-Codes 133 bis 140 (vergl. Liste der ASCII-Codes). Die Abfrage des Tastaturpuffers kann in etwa in Form einer GET-Schleife geschehen, wie im folgenden Programm gezeigt wird:

```
1 REM FUNKTIONSTASTEN
10 GETA$:IFA$=""THEN 10
20 IFA$=CHR$(133)THEN 40
30 GOTO 10
40 PRINT"F1-TASTE WURDE GEDRUECKT"
50 GOTO 10
READY.
```

Anhang H:

Organisation der zero page



Speicherbelegung im C64-Modus

(★ =nützliche Adressen)

	Hex	Dezimal	Beschreibung
	0000	0	6510 Daten Richtungsregister
	0001	1	6510 Ausgaberegister
	0002	2	nicht benutzt
	0003 – 0004	3 – 4	Vektor zur Umrechnung Float – Fixed
	0005 – 0006	5 – 6	Vektor zur Umrechnung Fixed – Float
	0007	7	Suchzeichen
	0008	8	Flag für Gänsefüßchen Modus
	0009	9	TAB Spaltenzähler
	000A	10	0 = LOAD, 1 = VERIFY
	000B	11	Zeiger für Eingabepuffer / Anzahl Elemente
	000C	12	Flag für Standard DIM
	000D	13	Typ: FF = String, 00 = numerisch
	000E	14	Typ: 80 = Integer, 00 = Fließkomma
	000F	15	Flag für DATA / LIST
	0010	16	Element / FNx Flag
	0011	17	00 = INPUT, 40 = GET, 98 = READ
	0012	18	Vorzeichen des ATN Flag für Gleichheit bei Vergleich
	0013	19	aktuelles E/A Gerät
★	0014 – 0015	20 – 21	Integer Wert
	0016	22	Zeiger auf temporären Stringstapel
	0017 – 0018	23 – 24	Letzter temporärer String Vektor
	0019 – 0021	25 – 33	Stapel für temporäre Strings
	0022 – 0025	34 – 37	Bereich für Hilfszeiger
	0026 – 002A	38 – 42	Bereich für Produkt bei Multiplikation
★	002B – 002C	43 – 44	Zeiger auf Basic Anfang
★	002D – 002E	45 – 46	Zeiger auf Variablen Anfang
★	002F – 0030	47 – 48	Zeiger auf Beginn der Arrays
★	0031 – 0032	49 – 50	Zeiger auf Ende der Arrays
★	0033 – 0034	51 – 52	Zeiger auf Stringspeicher (bewegt sich abwärts)
	0035 – 0036	53 – 54	Hilfszeiger für Strings
★	0037 – 0038	55 – 56	Zeiger auf Speichergrenze
	0039 – 003A	57 – 58	Nummer der aktuellen Basic Zeile

	Hex	Dezimal	Beschreibung
	003B – 003C	59 – 60	Nummer der vorhergehenden Basic Zeile
	003D – 003E	61 – 62	Zeiger auf Basic Statement für CONT
	003F – 0040	63 – 64	Nummer der aktuellen DATA Zeile
	0041 – 0042	65 – 66	Adresse des aktuellen DATA Elements
★	0043 – 0044	67 – 68	Sprungvektor für Input
	0045 – 0046	69 – 70	aktueller Variablenname
	0047 – 0048	71 – 72	Adresse der aktuellen Variablen
	0049 – 004A	73 – 74	Variablenzeiger für FOR . . . NEXT
	004B – 004C	75 – 76	Zwischenspeicher für Basic Zeiger
	004D	77	Akkumulator für Vergleichssymoble
	004E – 0053	78 – 83	verschieden genutzter Arbeitsbereich (Zeiger, usw.)
	0054 – 0056	84 – 86	Sprungvektor für Funktionen
	0057 – 0060	87 – 96	verschieden genutzter Bereich für numerische Operationen
★	0061	97	Fließkomma Akkumulator #1 (FAC): Exponent
★	0062 – 0065	98 – 101	Fließkomma Akkumulator #1 (FAC): Mantisse
★	0066	102	Fließkomma Akkumulator #1 (FAC): Vorzeichen
	0067	103	Zeiger für Polynom Auswertung
	0069 – 006E	105 – 110	Fließkomma Akkumulator #2 Exponent usw.
	006F	111	Vorzeichen Vergleich Akku #1 / Akku #2
	0070	112	niederwertige Stelle Akku #1 (Rundung)
	0071 – 0072	113 – 114	Länge des Kassettenpuffers
★	0073 – 008A	115 – 138	CHRGET Subroutine (hole ein Zeichen)
	007A – 007B	122 – 123	Basic Zeiger innerhalb der Subroutine
	008B – 008F	139 – 143	Startwert des Zufallgenerators
★	0090	144	Statusbyte ST
	0091	145	Flag für STOP und RVS Taste
	0092	146	Zeitkonstante für Kassette
	0093	147	0 = LOAD, 1 = VERIFY
	0094	148	serieller Ausgang: Flag für zurückgestelltes Zeichen

	Hex	Dezimal	Beschreibung
	0095	149	zurückgestelltes Zeichen
	0096	150	EOT von Kassette empfangen (Cassette Sync #)
	0097	151	Speicher für Register
★	0098	152	Anzahl offener Files
★	0099	153	Eingabegerät (normal = 0)
★	009A	154	Ausgabe (CMD) Gerät (normal = 3)
	009B	155	Paritätsbyte von Band
	009C	156	Flag für Byte empfangen
	009D	157	Ausgabe Kontrolle (80 = direkt, 00 = RUN)
	009E	158	Fehler vom Band / Zeichenpuffer
	009F	159	Fehler vom Band korrigiert
★	00A0 – 00A2	160 – 162	interne Uhr (HML)
	00A3	163	serieller Bit Zähler Flag für EOI
	00A4	164	Zyklen Zähler
	00A5	165	Abwärtszähler beim Schreiben auf Kassette
	00A6	166	Zeiger auf Kassettenpuffer
	00A7 – 00AB	167 – 171	Flags für Schreiben und Lesen von Kassette
	00AC – 00AD	172 – 173	Zeiger für Prg.Start
	00AE – 00AF	174 – 175	Zeiger für Programmende
	00B0 – 00B1	176 – 177	Zeitkonstanten für Band
★	00B2 – 00B3	178 – 179	Zeiger auf Anfang des Kassettenpuffers
	00B4	180	Band Timer (1 = gesetzt), Bit Zähler
	00B5	181	Band EOT / RS 232 nächstes Bit zum Senden
	00B6	182	★★★
★	00B7	183	Anzahl Zeichen im Filenamen
★	00B8	184	aktuelle logische Filenummer
★	00B9	185	aktuelle Sekundäradresse
★	00BA	186	aktuelles Gerät
★	00BB – 00BC	187 – 188	Zeiger auf Filenamen
	00BD	189	★★★
	00BE	190	Anzahl zum Lesen/Schreiben verbleibender Blocks

	Hex	Dezimal	Beschreibung
	00BF	191	serieller Wortpuffer
	00C0	192	Kassettenmotor Flag
	00C1 – 00C2	193 – 194	E/A Startadresse
	00C3 – 00C4	195 – 196	Zeiger auf Vektoradressen des KERNAL
★	00C6	198	Anzahl Zeichen im Tastaturpuffer
★	00C7	199	RVS Flag für Bildschirm
	00C8	200	Zeiger auf Zeilenende für Eingabe
	00C9 – 00CA	201 – 202	Position des Eingabecursors (Zeile, Spalte)
★	00CB	203	gedrückte Taste (64=keine Taste)
	00CC	204	Cursor an/aus (0=Cursor blinkt)
	00CD	205	Zähler für blinkenden Cursor
	00CE	206	Zeichen in Cursorposition
	00CF	207	Cursor in Blinkphase
	00D0	208	Eingabe von Bildschirm/Tastatur
★	00D1 – 00D2	209 – 210	Zeiger auf Bildschirmzeile
★	00D3	211	Zeiger auf Bildschirmspalte
	00D4	212	0=direkter Cursor, sonst programmiert (QUOTE MODE)
★	00D5	213	Länge der aktuellen Bildschirmzeile (40/80)
★	00D6	214	Zeile, in der sich der Cursor befindet
	00D7	215	Letzte Taste/Prüfsumme/Puffer
★	00D8	216	Anzahl ausstehender Inserts
★	00D9 – 00F0	217 – 240	Bildschirmzeilen Verknüpfungstabelle
	00F1	241	unechte Bildschirmzeile
	00F2	242	Bildschirmzeilen Marke
★	00F3 – 00F4	242 – 244	Zeiger auf Bildschirm Farbe
	00F5 – 00F6	245 – 246	Zeiger auf Tastatur Decodiertabelle
	00F7 – 00F8	247 – 248	RS232 Empfangszeiger
	00F9 – 00FA	249 – 250	RS232 Übertragungszeiger
★	00FB – 00FE	251 – 254	Freier Platz in Page 0 für Betriebssystem
	00FF	255	Basic Speicher
	0100 – 010A	256 – 266	Arbeitsbereich zur Umwandlung von Fließkomma in ASCII
	0100 – 013E	256 – 318	Bandfehler
	0100 – 01FF	256 – 511	Bereich des Prozessor Stapels
★	0200 – 0258	512 – 600	Basic Eingabepuffer

	Hex	Dezimal	Beschreibung	
★	0259 – 0262	601 – 610	Tabelle der logischen Files	
★	0263 – 026C	611 – 620	Tabelle der Gerätenummern	
★	026D – 0276	621 – 630	Tabelle der Sekundäradressen	
★	0277 – 0280	631 – 640	Tastaturpuffer	
★	0281 – 0282	641 – 642	Startadresse des RAM für Betriebssystem	
★	0283 – 0284	642 – 644	Ende des RAM für Betriebssystem	
	0285	645	Flag für Zeitüberschreitung auf seriellem Bus	
★	0286	646	aktueller Farbcode	
	0287	647	Farbe unter Cursor	
★	0288	648	Bildschirmspeicher Adresse (Page)	
★	0289	649	maximale Größe des Tastaturpuffers	
★	028A	650	Tastenwiederholung (128=alle Tasten)	
★	028B	651	Zähler für Wiederholungsgeschwindigkeit	
	028C	652	Zähler für Wiederholungsverzögerung	
★	028D	653	Flag für SHIFT/CNTRL	
	028E	654	letztes SHIFT Muster der Tastatur	
	028F – 0290	655 – 656	Zeiger auf Tastatur Decodiertabelle	
★	0291	657	SHIFT Modus (0=gesetzt, 128=blokkiert)	
	0292	658	automat. Scrolling abwärts (0=ein ≠0=aus)	
	0293	659	RS232 Kontrollregister	
	0294	660	RS232 Befehlsregister	
	0285 – 0296	661 – 662	nicht Standard (Bit Zeit) ★★★	
	0297	663	RS232 Statusregister	
	0298	664	Anzahl zu sendender Bits	
	0299 – 029A	665 – 666	Baud Rate	
	029B	667	RS232 Empfangszeiger	receive
	029C	668	RS232 Eingabezeiger	
	029D	669	RS232 Übertragungszeiger	send
	029E	670	RS232 Ausgabezeiger	
	029F – 02A0	271 – 672	enthält IRQ-Vektor während Kassettenbetrieb	
	02A1 – 02FF	673 – 767	★★★	
★	0300 – 0301	768 – 769	Vector für Fehlermeldungen	

Hex	Dezimal	Beschreibung
0302 – 0303	770 – 771	Vector für Basic Warmstart
0304 – 0305	772 – 773	Umwandlung von Schlüsselwörtern in Tokens
0306 – 0307	774 – 775	Umwandlung der Tokens in Schlüsselwörter
0308 – 0309	776 – 777	neuen Basic Befehl ausführen
030A – 030B	778 – 779	arithmetisches Element holen
030C	780	Speicher für 6502 „A Register
030D	781	Speicher für 6502 „X Register
030E	782	Speicher für 6502 „Y Register
030F	783	Speicher für 6502 „P Register
0310 – 0313	784 – 787	USR Sprung
0314 – 0315	788 – 789	Hardware Interrupt (IRQ) (EA31)
0316 – 0317	790 – 791	Break Interrupt (FE66)
0318 – 0319	792 – 793	nicht maskierbarer Interrupt (NMI) (FE47)
031A – 031B	794 – 795	OPEN (F40A) (F34A)
031C – 031D	796 – 797	CLOSE (F291)
031E – 031F	798 – 799	Kanal für Eingabe (F2C7) (F209)
0320 – 0321	800 – 801	Kanal für Ausgabe (F250)
0322 – 0323	802 – 803	Wiederherstellung der E/A (Löschen aller offenen Kanäle (F333)
0324 – 0325	804 – 805	INPUT (F157)
0326 – 0327	806 – 807	OUTPUT (F1CA)
0328 – 0329	808 – 809	STOP-Taste prüfen (F770) (F6ED)
032A – 032B	810 – 811	GET (F13E)
032C – 032D	812 – 813	Close aller Kanäle (F32F)
032E – 032F	814 – 815	Benutzer-IRQ (FE66)
0330 – 0331	816 – 817	RAM laden (F4A5)
0332 – 0333	818 – 819	RAM speichern (F5ED)
0334 – 033B	820 – 827	★★★
033C – 03FB	828 – 1019	Kassettenpuffer
0400 – 07FF	1024 – 2047	1K Bildschirmspeicher
(0400 – 07E7	1024 – 2023	Video Matrix)
(07F8 – 07FF	2040 – 2047	Zeiger für Sprites)
0800 – 9FFF	2048 – 40959	Basic Benutzerspeicher
A000 – BFFF	40960 – 49151	8K Basic ROM
C000 – CFFF	49152 – 53247	4K RAM

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
0000	00000	D8502	8502 Datenrichtungs-register
0001	00001	R8502	8502 Datenregister
0002	00002	BANK	Monitor und Long-Call/Jump-Register
0003	00003	PC-HI	Register für Befehlszähler High
0004	00004	PC-LO	Register für Befehlszähler Low
0005	00005	S-REG	Register für Statusbyte
0006	00006	A-REG	Register für Akkumulator
0007	00007	X-REG	X-Register
0008	00008	Y-REG	Y-Register
0009	00009	STKPTR	Stapelzeiger

Basic Zero-Page

Hex.	Dez.	Label	Bedeutung
0009	00009	INTEGR	Temporärer Speicher für den Integer-Wert bei OR/AND
0009	00009	CHARAC	Suchzeichen
000A	00010	ENDCHR	Flag: Suchen nach einem Ausführungszeichen am Ende eines Strings
000B	00011	TRMPOS	Bildschirmspalte ab letztem TAB
000C	00012	VERCK	Flag: 0 = LOAD, 1 = VERIFY
000D	00013	COUNT	Eingabepuffer, Anzahl der Elemente
000E	00014	DIMFLG	Flag: Standard-Felddimensionierung
000F	00015	VALTYP	Datentyp: $FF = String, $00 = numerisch
0010	00016	INTFLG	Datentyp: $80 = Ganze Zahl, $00 = Gleitkommazahl
0011	00017	GARBFL	Flag: DATAs lesen, LIST, Garbage-Collection
0011	00017	DORES	
0012	00018	SUBFLG	Flag: Benutzerfunktionsaufruf
0013	00019	INPFLG	Flag: $00 = INPUT, $40 = GET, $98 = READ
0014	00020	DOMASK	

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
0014	00020	TANSGN	Flag: Vorzeichen des TAN/Flag für Gleichheit bei Vergleich
0015	00021	CHANNL	Momentanes I/0-Gerät
0015	00021	POKER	Poke-Adresse
0016	00022	LINNUM	Zeilennummer Integer, oder 2-Byte-Adresse für GOTO,GOSUB,POKE,PEEK, SYS,WAIT
0018	00024	TEMPPT	Zeiger: Temporärer Stringstapel
0019	00025	LASTPT	Letzte Stringadresse
001B	00027	TEMPST	Stapel für temporäre Strings
0024	00036	INDEX	Bereich für Hilfszeiger
0024	00036	INDEX1	
0024	00036	INDEX2	
0028	00040	RESHO	Gleitkomma-Ergebnis der Multiplikation
0029	00041	RESMOH	
002A	00042	ADDEND	
002A	00042	RESMO	
002B	00043	RESLO	
002D	00045	TXTTAB	Zeiger: Anfang Basic-Text
002F	00047	VARTAB	Zeiger: Anfang Basic-Variablen
0031	00049	ARYTAB	Zeiger: Anfang Basic-Felder
0033	00051	STREND	Zeiger: Ende Basic-Felder + 1
0035	00053	FRETOP	Zeiger: Anfang der Stringspeicherung
0037	00055	FRESPC	Hilfszeiger für Strings
0039	00057	MAX-MEM-1	Oberste String/Variablen Adresse in Bank 1
003B	00059	CURLIN	Derzeitige Basic-Zeilennummer
003D	00061	TXTPTR	Zeiger auf Basic-Text für CHRGET, etc.
003F	00063	FORM	Wird von PRINT USING benutzt
003F	00063	FNDPNT	Zeiger auf gefundenes Byte (Von SEARCH)
0041	00065	DATLIN	Derzeitige DATA-Zeilennummer
0043	00067	DATPTR	Derzeitige DATA-Adresse

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
0045	00069	INPPTR	Vektor: INPUT-Routine
0047	00071	VARNAM	Derzeitiger Basic-Variablenname
0049	00073	FDECPT	
0049	00073	VARPNT	Adresse der aktuellen Variablen
004B	00075	LSTPNT	
004B	00075	ANDMSK	
004B	00075	FORPNT	Variablenzeiger für FOR/NEXT
004B	00075	EORMSK	
004D	00077	VARTXT	
004D	00077	OPPTR	Zwischenspeicher für Basic-Zeiger/Daten
004F	00079	OPMASK	Vergleichs-Maske, größer = $01, gleich = $02, kleiner = $04
0050	00080	GRBPNT	
0050	00080	TEMPF3	
0050	00080	DEFPNT	Zeiger auf Variable einer DEF FN-Funktion
0052	00082	DSCPNT	Zeiger auf String-Diskriptor in einer Variablen-Liste
0055	00085	HELPER	Flag: HELP oder LIST
0056	00086	JMPER	
0058	00088	OLDOV	
0059	00089	TEMPF1	Temporärer Zeiger, temporärer Gleitpunktakkumulator
005A	00090	ARYPNT	Zeiger zur Initialisierung bei einem DIM-Befehl
005A	00090	HIGHDS	
005C	00092	HIGHTR	Zeiger, der von der Block Transfer Routine benutzt wird
005E	00094	TEMPF2	
005F	00095	DECCNT	
0061	00097	GRBTOP	
0061	00097	DPTFLG	Wird auf $80 gesetzt, wenn ein Eingabestring (Zahl) einen Dezimalpunkt enthält
0061	00097	LOWTR	

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
0062	00098	EXPSGN	Vorzeichen des Exponenten, $80 = negativ
0063	00099	FAC1	Gleitkommaakkumulator #1, besteht aus dem Exponenten, 4 Bytes für die Mantisse, und einem Vorzeichen-Byte. Integer-Ergebnisse werden in FAC1 + 3 und FAC1 + 4 abgespeichert
0069	00105	DEGREE	
0069	00105	SGNFLG	Zeiger für Polynomauswertung
006A	00106	ARGEXP	Gleitkommaakkumulator #2, Exponat, 4 Bytes Mantisse, Vorzeichenbyte
0070	00112	STRNG1	
0070	00112	ARISGN	Vorzeichenvergleich Akku #1 mit Akku #2. $00 = gleiches Vorzeichen, $FF = unterschiedlich
0071	00113	FACOV	Akku #1, niederwertige Stelle, Rundung
0072	00114	STRNG2	
0072	00114	POLYPT	
0072	00114	CURTOL	
0072	00114	FBUFPT	Zeiger: Kassettenpuffer
0074	00116	AUTINC	Inkrement beim AUTO-Befehl, $00 = aus
0076	00118	MVDFLG	Flag: Gesetzt, wenn 10KByte für Hires-Grafik reserviert
0077	00119	NOZE	Anzahl führender Nullen beim PRINT USING-Befehl
0077	00119	SPRNUM	Temporärer Speicher bei SPRITE- und MOVSPR-Befehlen
0077	00119	KEYNUM	Temprärer Speicher für MID$-Befehl
0078	00120	HULP	Zähler
0078	00120	KEYSIZ	
0079	00121	SYNTMP	Temporärer Speicher bei indirektem Laden

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
007A	00122	DSDESC	Deskriptor für DS$
007D	00125	TOS	Obergrenze des Run-Time-Stack
007F	00127	RUNMOD	Flag: RUN/DIRECT-Modus
0080	00128	PARSTS	DOS Statuswort, Syntaxcheck
0080	00128	POINT	Zeiger bei PRINT USING auf Dezimalpunkt
0081	00129	PARSTX	
0082	00130	OLDSTK	

Zero-Page für Grafik

Hex.	Dez.	Label	Bedeutung
0083	00131	COLSEL	Ausgewählte aktuelle Farbe
0084	00132	MULTICOLOR-1	
0085	00133	MULTICOLOR-2	
0086	00134	FOREGROUND	Vordergrundfarbe
0087	00135	SCALE-X	SCALE-Faktor in X-Richtung
0089	00137	SCALE-Y	SCALE-Faktor in Y-Richtung
008B	00139	STOPNB	Vergleichsregister für PAINT
008C	00140	GRAPNT	
008E	00142	VTEMP1	
008F	00143	VTEMP2	

Zero-Page für Kernal/Editor

Hex.	Dez.	Label	Bedeutung
0090	00144	STATUS	Statusbyte bei I/O-Operationen
0091	00145	STKEY	Flag: STOP-Taste
0092	00146	SVXT	Zeit-Konstante für Kassette
0093	00147	VERCK	Flag: $00 = LOAD, $01 = VERIFY
0094	00148	C3PO	Flag: Serieller Bus-Zeichen im Buffer
0095	00149	BSOUR	Zeichen im Puffer für seriellen Bus
0096	00150	SYNO	Kassetten SYNC.-Nr. (EOT von Kassette empfangen)
0097	00151	XSAV	Temporäre Datenadresse
0098	00152	LDTND	Anzahl der offenen Dateien/Dateitabellen-Index
0099	00153	DFLTN	Standard-Eingabegerät
009A	00154	DFLTO	Standard-Ausgabegerät

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
009B	00155	PRTY	Paritätsbyte von Kassette
009C	00156	DPSW	Flag: Byte empfangen
009D	00157	MSGFLG	Flag: $80 = Direktmodus, $00 = Programm
009E	00158	PTR1	Bandfehler/Zeichenpuffer
009E	00158	T1	Temporärer Speicher
009F	00159	PTR2	Bandfehler korrigiert
009F	00159	T2	Temporärer Speicher
00A0	00160	TIME	Echtzeituhr (ca.) 1/60 s
00A3	00163	R2D2	Temporärer Datenbereich
00A3	00163	PCNTR	Temporärer Speicher (Kassette)
00A4	00164	BSOUR1	Temporärer Speicher (Serielle-Routine)
00A5	00165	COUNT	Temporärer Speicher (Serielle-Routine)
00A5	00165	CNTDN	Kassetten Sync.: Abwärtszählung beim Schreiben
00A6	00166	BUFPT	Zeiger: Kassettenpuffer
00A7	00167	INBIT	RS-232-Eingabebits/Kassette temporär
00A8	00168	BITCI	RS-232-Eingabebit-Zählung/Kassette temporär
00A9	00169	RINONE	RS-232-Startbit-Überprüfung
00AA	00170	RIDATA	RS-232-Eingabebyte-Puffer/Kassette temporär
00AB	00171	RIPRTY	RS-232-Eingabe-parität/Kassette, Zählung
00AC	00172	SAL	Zeiger: Kassettenpuffer/ Bildschirm scrollen
00AE	00174	EAL	Kassettenende/Programmende
00B0	00176	CMPO	Kassetten-Zeit-Konstante
00B2	00178	TAPE1	Zeiger: Anfang des Kassettenpuffers
00B4	00180	BITTS	RS-232-Bitzähler (Senden)
00B5	00181	NXTBIT	RS-232-Nächstes zu übertragendes Bit
00B6	00182	RODATA	RS-232-Bytepuffer
00B7	00183	FNLEN	Länge des aktuellen Dateinamens

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
00B8	00184	LA	Logische Dateinummer
00B9	00185	SA	Aktuelle Sekundäradresse
00BA	00186	FA	Aktuelle Gerätenummer
00BB	00187	FNADR	Zeiger: Aktueller Dateiname
00BD	00189	ROPRTY	RS-232-Parität/Kassette temporär
00BE	00190	FSBLK	Anzahl der zum Lesen/Schreiben verbleibenden Blocks (Kassette)
00BF	00191	DRIVE	
00C0	00192	CAS1	Kassettenmotor-Flag
00C1	00193	STAL	Ein-/Ausgabe-Startadresse (Low)
00C2	00194	STAH	Ein-/Ausgabe-Startadresse (High)
00C3	00195	MEMUSS	Temporärer Speicher (Zeiger)
00C5	00197	DATA	Kassette: Schreib-/Lese-Daten
00C6	00198	BA	Aktuelle Bank für LOAD/SAVE/VERIFY-Operationen
00C7	00199	FNBANK	Bank, in der sich der aktuelle Filenamen befindet
00C8	00200	RIBUF	RS-232-Eingabepuffer-Zeiger
00CA	00202	ROBUF	RS-232-Ausgabepuffer-Zeiger

Zero-Page für 40/80-Zeichen Editor

Hex.	Dez.	Label	Bedeutung
00CC	00204	KEYTAB	Vektor: Tastatur-Decodiertabelle
00CE	00206	IMPARM	Hilfszeiger für Strings
00D0	00208	NDX	Anzahl der Zeichen im Tastaturpuffer (Warteschlange)
00D1	00209	KYNDX	Zähler für Zeichen einer Funktionstaste
00D2	00210	KEYIDX	Zeiger auf String einer Funktionstaste
00D3	00211	SHFLAG	Flag für: SHIFT/CONTROL/COMMODORE-Taste
00D4	00212	SFDX	Nummer der augenblicklich gedrückten Taste
00D5	00213	LSTX	Nummer der zuletzt gedrückten Taste

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
00D6	00214	CRSW	RETURN-Input-Flag
00D7	00215	MODE	40/80-Zeichen-Flag $00 = 40- $80 = 80-Zeichen
00D8	00216	GRAPHM	Text/Grafik-Flag
00D9	00217	CHAREN	Ram/Rom-Flag für Video-Chip (40-Zeichen), Bit 2
00DA	00218	SEDSAL	Bildschirmeditor: Startadresse Zeile
00DC	00220	SEDEAL	Bildschirmeditor: Endadresse Zeile
00DE	00222	SEDT1	Temporärer Speicher für Editor
00DF	00223	SEDT2	Temporärer Speicher für Editor
00DA	00218	KEYSIZ	Register für Funktionstasten
00DB	00219	KEYLEN	
00DC	00220	KEYNUM	
00DD	00221	KEYNXT	
00DE	00222	KEYBNK	
00DF	00223	KEYTMP	
00DA	00218	BITMSK	Temporäres Register für TAB und Zeilenüberlauf (Line Wrap)
00DB	00219	SAVER	Temporäres Register

C 128 Systemadressen

Die folgenden Adressen beziehen sich auf den gerade aktuellen Bildschirm (40- oder 80-Zeichen)
und werden nach Bedarf in einen anderen Speicherbereich ($0A40) kopiert. Damit bleiben die jeweilige Cursorposition, Fensterdefinition u. dgl. auch bei einem Wechsel von 40- auf 80-Zeichen, oder umgekehrt, erhalten.

Adresse: Hex.	Dez.	Label	Bedeutung
00E0	00224	PNT	Zeiger auf aktuelle Zeile (Text)
00E2	00226	USER	Zeiger auf aktuelle Zeile (Attribute)
00E4	00228	SCBOT	Fenster-Untergrenze
00E5	00229	SCTOP	Fenster-Obergrenze
00E6	00230	SCLF	Fenster-linker Rand
00E7	00231	SCRT	Fenster-rechter Rand
00E8	00232	LSXP	Aktuelle Eingabe-Spalte, Start
00E9	00233	LSTP	Aktuelle Eingabe-Zeile, Start
00EA	00234	INDX	Aktuelle Eingabe-Zeile, Ende
00EB	00235	TBLX	Aktuelle Cursor-Zeile
00EC	00236	PNTR	Aktuelle Cursor-Spalte
00ED	00237	LINES	Maximale Anzahl der Zeilen
00EE	00238	COLUMNS	Maximale Anzahl der Bildschirmfarben
00EF	00239	DATAX	Nächstes auszugebendes Zeichen
00F0	00240	LSTCHR	Vorhergehendes Zeichen (für ESC-Test)
00F1	00241	COLOR	Attribut des nächsten auszugebenden Zeichens (Standard: Vordergrundfarbe)
00F2	00242	TCOLOR	Temporäres Register für die Farbe des Zeichens (Insert & Delete)
00F3	00243	RVS	Reverse-Flag
00F4	00244	QTSW	Anführungszeichen-Flag
00F5	00245	INSRT	Einfügemodus-Flag
00F6	00246	INSFLG	Flag für automatisches Einfügen
00F7	00247	LOCKS	Verhindert Commodore/Shift und Ctrl-S
00F8	00248	SCROLL	Verhindert Bildschirmscrollen und Zeilenverknüpfung
00F9	00249	BEEPER	Verhindert Ctrl-G (Bell)
00FF	00255	LOFBUF	

C 128 Systemadressen

Die folgenden Adressen werden vom MONITOR benutzt

Adresse: Hex.	Dez.	Label	Bedeutung
0002	00002	PCB	
0003	00003	PCH	
0004	00004	PCL	
0005	00005	FLGS	
0006	00006	ACC	
0007	00007	XR	
0008	00008	YR	
0009	00009	SP	
007A	00122	TXTPTR	

Basis/DOS-Bereich

Adresse: Hex.	Dez.	Label	Bedeutung
0100	00256	FBUFFR	Bereich für den Filenamen (16-Zeichen)
0110	00272	XCNT	Zähler für das DOS
0111	00273	DOSF1L	Länge des Filenamens 1
0112	00274	DOSDS1	Laufwerk 1
0113	00275	DOSF2L	Länge des Filenamens 2
0114	00276	DOSDS2	Laufwerk 2
0115	00277	DOSF2A	Adresse des Filenamens 2
0117	00279	DOSOFL	BLOAD/BSAVE Startadresse
0119	00281	DOSOFH	BSAVE Endadresse
011B	00283	DOSLA	Logische Adresse
011C	00284	DOSFA	Physikalische Adresse
011D	00285	DOSSA	Sektoradresse
011E	00286	DOSRCL	Record Länge
011F	00287	DOSBNK	
0120	00288	DOSDID	Laufwerks Identifizierung
0122	00290	DIDCHK	Flag: Arbeit beendet

C 128 Systemadressen

Die folgenden Adressen werden bei PRINT USING benutzt

Adresse: Hex.	Dez.	Label	Bedeutung
0123	00291	BNR	Zeiger auf Beginn
0124	00292	ENR	Zeiger auf Ende
0125	00293	DOLR	Dollar Flag
0126	00294	FLAG	Komma Flag
0127	00295	SWE	Zähler
0128	00296	USGN	Vorzeichen Exponent
0129	00297	UEXP	Zeiger auf Exponent
012A	00298	VN	Anzahl der Zahlen vor dem Dezimalpunkt
012B	00299	CHSN	Justierungs-Flag
012C	00300	VF	Anzahl der Positionen vor dem Dezimalpunkt
012D	00301	NF	Anzahl der Positionen nach dem Dezimalpunkt
012E	00302	POSP	+/– Flag (Feld)
012F	00303	FESP	Exponent-Flag (Feld)
0130	00304	ETOF	Schalter
0131	00305	CFORM	Zeichenzähler (Feld)
0132	00306	SNO	Vorzeichennummer
0133	00307	BLFD	Space/Stern-Flag
0134	00308	BEGFD	Zeiger auf Anfang des Feldes
0135	00309	LFOR	Länge des Formatstrings
0136	00310	ENDFD	Zeiger auf Ende des Feldes
0137	00311	SYSSTK	System-Stack
0200	00512	BUF	Basic/Monitor-Puffer
02FC	00764	ESC-FN-VEC	Vektor für zusätzliche Funktions-Routinen
02FE	00766	BNKVEC	Vektor für Function-Cartridge

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
Indirekte RAM-Zeiger (Basic)			
0300	00768	IERROR	Fehler Routine (Fehler in X)
0302	00770	IMAIN	Vektor: Basic-Warmstart
0304	00772	ICRNCH	Token-Umwandlungsroutine
0306	00774	IQPLOP	Vektor: Basic-Text listen
0308	00776	IGONE	Vektor: Basic-Befehl ausführen
030A	00778	IEVAL	Token auswerten
030C	00780	IESCLK	Escape-Umwandlungsroutine
030E	00782	IESCPR	Escape List
0310	00784	IESCEX	Escape ausführen
Indirekte Kernal-Vektoren			
0312	00786	ITIME	Interruptvektor TIME
0314	00788	IIRQ	IRQ Ram-Vektor
0316	00790	IBRK	BRK-Ram-Vektor
0318	00792	INMI	NMI-Ram-Vektor
031A	00794	IOPEN	
031C	00796	ICLOSE	
031E	00798	ICHKIN	
0320	00800	ICHKOUT	
0322	00802	ICLRCH	
0324	00804	IBASIN	
0326	00806	IBASOUT	
0328	00808	ISTOP	
032A	00810	IGETIN	
032C	00812	ICLALL	
032E	00814	EXMON	Indirekter Sprung MONITOR
0330	00816	ILOAD	
0332	00818	ISAVE	

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
Unterprogramme für indirektes Laden (Aus jeder Bank)			
039F	00927	IND-SUB-RAMO	Unterprogramm zum Laden aus beliebiger Bank (PCRA und PCRC vorbesetzen)
03AB	00939	IND-SUB-RAM1	S.o. PCRB und PCRD vorbesetzen
03B7	00951	INDIN1-RAM1	S.o. Zero-Page-Indirekt $24
03C0	00960	INDIN2	S.o. PCRA und PCRC $26
03C9	00969	INDTXT	S.o. $3D
03D2	00978	ZERO	Konstante für Basic
03D5	00981	CURRENT BANK	Register für Bank bei PEEK, SYS, POKE Befehlen
03D6	00982	TMPDES	Temporäre Register
03DA	00986	FIN-BANK	Bank-Zeiger für Zahl/String-Umwandlung
03DB	00987	SAVSIZ	Temporärer Speicher für SSHAPE
03DF	00991	BITS	
03E0	00992	SPRTMP-1	Temporärer Speicher für SPRSAV
03E1	00993	SPRTMP-2	
03E2	00994	FG-BG	Vordergrund/Hintergrund Farb-Nibble (Gepackt)
03E3	00995	FG-MC1	Vordergrund/Multicolor 1 Nibble (Gepackt)
0400	01024	VICSCN	Bildschirmspeicher 40-Zeichen
0800	02048		Basic-Run-Time-Stack (512 Bytes)
0A00	02560	SYSTEM	Restart Vektor (Basic Warmstart)
0A02	02562	DEJAVU	Kernal Warm/Kalt-Start Statusbyte
0A03	02563	PALNTS	PAL/NTSC System-Flag
0A04	02564	INIT-STATUS	Flag: Reset/Nmi Status

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
0A05	02565	MEMSTR	Zeiger auf Anfang des verfügbaren Speichers in der System-Bank
0A07	02567	MEMSIZ	S.o. Ende
0A09	02569	IRQTMP	Indirekter IRQ-Vektor (Kassette)
0A0B	02571	CASTON	Zeitvergleich bei Kassetten-Operationen
0A0C	02572	KIKA26	Temporärer Speicher beim Kassette-Lesen
0A0D	02573	STUPID	Temporärer Speicher Kassette-Lesen
0A0E	02574	TIMOUT	Timeout-Flag (Fast Serial)
0A0F	02575	ENABL	RS-232 Enable
0A10	02576	M51CTR	RS-232 Control Register
0A11	02577	M51CDR	RS-232 Command Register
0A12	02578	M51AJB	RS-232 Baud-Rate
0A14	02580	RSSTAT	RS-232 Status Register
0A15	02581	BITNUM	RS-232 Wortlänge
0A16	02582	BAUDOF	Zwischenspeicher Baud-Rate
0A18	02584	RIDBE	RS-232 Zeiger auf Ende des Eingabepuffers
0A19	02585	RIDBS	RS-232 Zeiger auf Start des Eingabepuffers
0A1A	02586	RODBS	RS-232 Zeiger auf Start des Ausgabepuffers
0A1B	02587	RODBE	RS-232 Zeiger auf Ende des Ausgabepuffers
0A1C	02588	SERIAL	Intern/Extern-Flag (Fast Serial)
0A1D	02589	TIMER	Zwischenspeicher Echtzeituhr
0A20	02592	XMAX	Größe Tastatur-Warteschlange
0A21	02593	PAUSE	Flag für Ctrl-S

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
VIC = 40-Zeichen-Video-Controller, VDC = 80-Zeichen-Video-Controller			
0A22	02594	RPTFLG	Ermöglicht Tasten-Wiederholung $80 = alle Tasten, $40 = keine Taste, $00 nur Cursortasten/DEL/INST
0A23	02595	KOUNT	Zähler für Verzögerung bei Tasten-Wiederholung
0A24	02596	DELAY	Zähler für Verzögerung bis zum Einsetzen der Tasten-Wiederholung
0A25	02597	LSTSHF	Zähler für Verzögerung bei Commodore-Shift
0A26	02598	BLNON	VIC: Cursor-Modus $40 = fester Cursor
0A27	02599	BLNSW	VIC: Hilfsregister
0A28	02600	BLNCT	VIC: Hilfsregister
0A29	02601	GDBLN	VIC: Cursorzeichen vor dem Blinken
0A2A	02602	GDCOL	VIC: Cursorfarbe vor dem Blinken
1000	04096	PKYBUF	Bereich für die programmierbaren Funktionstasten. Die ersten 10 Bytes geben jeweils die Länge der jeweiligen Strings an. Belegbar sind: F1-F8, Shift-Run, Help
100A	04106	PKYDEF	Bereich für die, den Funktionstasten zugeordneten Strings
1100	04352	DOSSTR	Bereich für DOS-String

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
Bereich für Grafik			
1131	04401	VWORK	
1131	04401	XYPOS	
1131	04401	XPOS	Aktuelle X-Position
1133	04403	YPOS	Aktuelle Y-Position
1135	04405	XDEST	X-Koordinate Ziel
1137	04407	YDEST	Y-Koordinate Ziel
Linie zeichnen			
1139	04409	XYABS	
1139	04409	XABS	
113B	04411	YABS	
113D	04413	XYSGN	
113D	04413	XSGN	
113F	04415	YSGN	
1141	04417	FCT	
1145	04421	ERRVAL	
1147	04423	LESSER	
1148	04424	GREATR	
Variablen für Winkel-Routinen			
1149	04425	ANGSGN	Vorzeichen Winkel
114A	04426	SINVAL	Sinus des Winkels
114C	04428	COSVAL	Cosinus des Winkels
114E	04430	ANGCNT	Temporäres Register für Winkelabstand
Variablen für CIRCLE-Befehl			
1150	04432	XCIRC1	Kreismitte, X-Koordinate
1152	04434	YCIRC1	Kreismitte, Y-Koordinate
1154	04436	XRADUS	X-Radius
1156	04438	YRADUS	Y-Radius

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
0A2B	02603	CURMOD	VDC: Cursormodus: $80 = fest, $60 = normal blinkend, $40 = schnell blinkend, $20 = Aus
0A2C	02604	VM1	VIC: Text-Basiszeiger
0A2D	02605	VM2	VIC: Bit-Map-Basiszeiger
0A2E	02606	VM3	VDC: Text-Basiszeiger
0A2F	02607	VM4	VDC: Attribute-Basiszeiger
0A30	02608	LINTMP	Temporärer Zeiger auf letzte Zeile
0A31	02609	SAV80A	VDC: Temporäres Register
0A32	02610	SAV80B	VDC: Temporäres Register
0A33	02611	SAV80C	VDC: Temporäres Register
0A34	02612	SAV80D	VDC: Temporäres Register
0A35	02613	CURCOL	VDC: Cursorfarbe vor dem Blinken
0A36	02614	SPLIT	VIC: Splitscreen Raster-Wert
0A37	02615	FNADRX	Zwischenspeicher während Bankoperationen
0A38	02616	PALCNT	Hilfszähler für Echtzeituhr bei PAL-System

Systemadressen, die vom MONITOR benutzt werden:

Hex.	Dez.	Label	Bedeutung
0A80	02688	XCNT	Puffervergleich
0AA0	02720	HULP	
0AAA	02730	FORMAT	
0AAB	02731	LENGTH	Disassembler/Assembler
0AAC	02732	MSAL	Für Assembler
0AAF	02734	SXREG	Temporär
0AB0	02735	SYREG	Temporär
0AB1	02736	WRAP	Temporär für Assembler
0AB2	02737	XSAVE	X-Register bei indirekten Unterprogramm-Aufrufen sichern
0AB3	02738	DIRECTION	Richtungsanzeige für Transfer
0AB4	02739	TEMPS	

Hex.	Dez.	Label	Bedeutung
0AC0	02752	CURBNK	Aktuelle Funktionstasten ROM-Bank
0AC1	02753	PAT	Physikalische Adressen (Tabelle)

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
0B00	02816	TBUFFR	Kassettenpuffer und Puffer für BOOT-Sektor
0C00	03072	RS2331	RS-232 Eingabepuffer
0D00	03328	RS2320	RS-232 Ausgabepuffer
0E00	03584		Bereich für Sprite-Definitionen ($E00-$FFF)
1158	04440	ROTANG	Drehwinkel
115C	04444	ANGBEG	Kreissegment-Winkel Start
115E	04446	ANGEND	Kreissegment-Winkel Ende
1160	04448	XRCOS	X-Radius * Cosinus des Drehwinkels
1162	04450	YRSIN	Y-Radius * Sinus des Drehwinkels
1164	04452	XRSIN	X-Radius * Sinus des Drehwinkels
1166	04454	YRCOS	Y-Radius * Cosinus des Drehwinkels

Allgemein benutzte Grafikvariablen (Mehrfach definiert)

Hex.	Dez.	Label	Bedeutung
1150	04432	XCENTR	
1152	04434	YCENTR	
1154	04436	XDIST1	
1156	04438	YDIST1	
1158	04440	XDIST2	
115A	04442	YDIST2	
115C	04444	DISEND	
115E	04446	COLCNT	Für CHAR-Befehl
115F	04447	ROWCNT	Für CHAR-Befehl
1160	04448	STRCNT	Für CHAR-Befehl

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
Variablen für BOX-Befehl			
1150	04432	XCORD1	Punkt 1 X-Koordinate
1152	04434	YCORD1	Punkt 1 Y-Koordinate
1154	04436	BOXANG	Drehwinkel
1156	04438	XCOUNT	
1158	04440	YCOUNT	
115A	04442	BXLENG	Länge einer Seite
115C	04444	XCORD2	Punkt 2 X-Koordinate
115E	04446	YCORD2	Punkt 2 Y-Koordinate
Variablen für SSHAPE/GSHAPE			
1151	04433	LEYLEN	
1152	04434	KEYNXT	
1153	04435	STRSZ	Länge der Stringvariablen
1154	04436	GETTYP	Shape-Modus setzen
1155	04437	STRPTR	Zähler für Stringposition
1156	04438	OLDBYT	Altes Bit-Mapped-Byte
1157	04439	NEWBYT	Variable für neuen String oder Bit-Mapped-Byte
1159	04441	XSIZE	Shape, Länge in X-Richtung
115B	04443	YSIZE	Shape, Länge in Y-Richtung
Variablen für SSHAPE/GSHAPE			
115D	04445	XSAVE	Temporärer Speicher (XSIZE)
115F	04447	STRADR	Speicher für Shape/String Deskriptor
1161	04449	BITIDX	Zeiger auf ein Bit in einem Byte

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
Grafikvariablen allgemein			
1168	04456	CHRPAG	High-Byte einer Character-Room Adresse für CHAR-Befehl
1169	04457	BITCNT	Register für GSHAPE
116A	04458	SCALEM	SCALE-Modus-Flag
116B	04459	WIDTH	Flag für doppelte Pixel-Größe
116C	04460	FILFLG	Flag für Ausmalen eines Rechtecks (BOX-Befehl)
116D	04461	BITMSK	Temporärer Speicher für Bitmaske
116E	04462	NUMCNT	
116F	04463	TRCFLGFlage für Trace-Modus	
1170	04464	RENUM-TMP-1	Temporärer Speicher für RENUMBER
1172	04466	RENUM-TMP-2	Temporärer Speicher für RENUMBER
1174	04468	T3	
1175	04469	T4	
1177	04471	VTEMP3	Temporärer Speicher für Grafik
1178	04472	VTEMP4	S.o.
1179	04473	VTEMP5	S.o.
117A	04474	ADRAY1	Zeiger auf Konvertierungsroutine Gleitkomma nach Integer
117C	04476	ADRAY2	S.o. Integer nach Fließkomma
117E	04478	SPRITE-DATA	Tabelle für Geschwindigkeit und Richtung der Sprites
11D6	04566	VIC-SAVE	Temporärer Speicher für VIC-Register
Allgemeiner Bereich für Basic			
1200	04608	OLDLIN	Vorhergehende Basic-Zeilennummer
1202	04610	OLDTXT	Zeiger auf Basicbefehl (Für CONTINUE)

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
Festlegungen für PRINT USING			
1204	04612	PUCHRS	
1204	04612	PUFILL	Füllzeichen bei PRINT-USING (Standard: Space)
1205	04613	PUCOMA	Kommasymbol
1206	04614	PUDOT	Dezimalpunkt-Symbol
1207	04615	PUMONY	Währungszeichen
1208	04616	ERRNUM	Letzte Fehlernummer
1209	04617	ERRLIN	Zeilennummer in der der letzte Fehler auftrat, $FFFF = kein Fehler
120B	04619	TRAPNO	Verweis auf Zeilennummer für ON ERROR GOTO
120D	04621	TMPTRP	Temporäres Register für TRAP
120E	04622	ERRTXT	
1210	04624	TEXT-TOP	Zeiger auf Ende Basic-Text
1212	04626	MAX-MEM-0	Höchste für Basic verfügbare Adresse in Bank 0
1214	04628	TMPTXT	Wird bei einer D0-Schleife benutzt
1216	04630	TMPLIN	
1218	04632	USRPOK	
121B	04634	RNDX	
1220	04640	CIRCLE-SEGMENT	Winkel pro Kreissegment
1221	04641	DEJAVU	Reset-Status (Kalt/Warm)
Variablen für Musikbefehle			
1222	04642	TEMPO-RATE	
1223	04643	VOICES	
1229	04649	NTIME	
122B	04651	OCTAVE	
122C	04652	SHARP	
122D	04653	PITCH	
122F	04655	VOICE	
1230	04656	WAVEO	
1233	04659	DNOTE	

C 128 Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
1234	04660	FLTSAV	
1238	04664	FLTFLG	
1239	04665	NIBBLE	
123A	04666	TONNUM	
123B	04667	TONVAL	
123E	04670	PARCNT	
123F	04671	ATKTAB	
1249	04681	SUSTAB	
1253	04691	WAVTAB	
125D	04701	PULSLW	
1267	04711	PULSHI	
1271	04721	FILTERS	
1276	04726	INT-TRIP-FLAG Interrupt Flag	
1279	04729	INT-ADR-LO Interrupt Flag	
127C	04732	INT-ADR-HI Interrupt Flag	
127F	04735	INTVAL	
1280	04736	COLTYP	

Variablen für SOUND-Befehl

Hex.	Dez.	Label
1281	04737	SOUND-VOICE
1282	04738	SOUND-TIME-LO
1285	04741	SOUND-TIME-HI
1288	04744	SOUND-MAX-LO
128B	04747	SOUND-MAX-HI
128E	04750	SOUND-MIN-LO
1291	04753	SOUND-MIN-HI
1294	04756	SOUND-DIRECTION
1297	04759	SOUND-STEP-LO
129A	04762	SOUND-STEP-HI
129D	04765	SOUND-FREQ-LO
12A0	04768	SOUND-FREQ-HI
12A3	04771	TEMP-TIME-LO
12A4	04772	TEMP-TIME-HI
12A5	04773	TEMP-MAX-LO
12A6	04774	TEMP-MAX-HI

C 128 Systemadressen

Adresse:

Hex.	Dez.	Label	Bedeutung
12A7	04775	TEMP-MIN-LO	
12A8	04776	TEMP-MIN-HI	
12A9	04777	TEMP-DIRECTION	
12AA	04778	TEMP-STEP-LO	
12AB	04779	TEMP-STEP-HI	
12AC	04780	TEMP-FREQ-LO	
12AD	04781	TEMP-FREQ-HI	
12AE	04782	TEMP-PULSE-LO	
12AF	04783	TEMP-PULSE-HI	
12B0	04784	TEMP-WAVEFORM	
12B1	04785	POT-TEMP-1	Temporäres Register für POT
12B2	04786	POT-TEMP-2	Temporäres Register für POT
12B3	04787	WINDOW-TEMP	Temporäres Register für Window-Befehl
12B7	04791	SAVRAM	Register für SPRDEF
12FA	04858	DEFMOD	Register für SPRDEF
12FB	04859	LINCNT	Register für SPRDEF
12FC	04860	SPRITE-NUMBER	Register für SPRDEF

Hex.	Dez.	Bedeutung
1300	04864	Unbenutzter Ram-Bereich ($1300-$17FF)
1800	06144	Reservierter Ram-Bereich ($1800-$BFF) für Function-Key-Software
1C00	07168	Video-Matrix, Bitmap-Color, wenn Grafik benutzt wird
1C00	07168	Start Basic-Text (Ohne Grafik, Standardwert MEMTOP)
2000	08192	VIC Bitmap (8KByte, wenn Grafik benutzt wird)
4000	16384	Start Basic-Text bei Grafik

C 128 I/0-Systemadressen

Adresse: Hex.	Dez.	Label	Bedeutung
D000	53248	VICCHR	Character Rom
D000	53248	VICREG	VIC Register
D400	54272	SIDREG	SID Register
D500	54528	MMU-LO	Register der MMU im I/0-Bereich
D600	54784	VDC	80-Zeichen Videochip (VDC)
D800	55296	VICCOL	VIC Color-Nibble
DC00	56320	CIA1	6526 1
DD00	56576	CIA2	6525 2
DE00	56832	I01	Expansion I/0-Slot
DF00	57088	I02	Expansion I/0-Slot (Reserviert für optimalen DMA-Controller)
FF00	65280	MMU-HI	Register der MMU im Common-Bereich

Anhang I:

BASIC-Abkürzungen



ABKÜRZUNGEN DER BASIC SCHLÜSSELWÖRTER

Damit Sie beim Eintippen Ihrer Programme und Direktkommandos Zeit sparen können, dürfen Sie die meisten Schlüsselwörter im BASIC 2.0 (C64-Modus) abkürzen. Die Abkürzung für PRINT ist das Fragezeichen. Die Abkürzungen für andere Schlüsselwörter werden gebildet, indem man die ersten ein oder zwei Zeichen eintippt und das nächste Zeichen des Wortes mit SHIFT eingibt. Werden die Abkürzungen in Programmzeilen benutzt, gibt LIST die Schlüsselwörter in der ausgeschriebenen Form aus. Beachten Sie, daß Abkürzungen für einige Schlüsselwörter bereits eine linke Klammer mit einschließen.

Befehl	Abkürzung	wird angezeigt als	Befehl	Abkürzung	wird angezeigt als
ABS	A SHIFT B	A	FOR	F SHIFT O	F
AND	A SHIFT N	A	FRE	F SHIFT R	F
ASC	A SHIFT S	A ♥	GET	G SHIFT E	G
ATN	A SHIFT T	A	GOSUB	GO SHIFT S	GO ♥
CHR$	C SHIFT H	C	GOTO	G SHIFT O	G
CLOSE	CL SHIFT O	CL	INPUT#	I SHIFT N	I
CLR	C SHIFT L	C	LET	L SHIFT E	L
CMD	C SHIFT M	C	LEFT$	LE SHIFT F	LE
CONT	C SHIFT O	C	LIST	L SHIFT I	L
DATA	D SHIFT A	D ♠	LOAD	L SHIFT O	L
DEF	D SHIFT E	D	MID$	M SHIFT I	M
DIM	D SHIFT I	D	NEXT	N SHIFT E	N
END	E SHIFT N	E	NOT	N SHIFT O	N
EXP	E SHIFT X	E ♣	OPEN	O SHIFT P	O

Befehl	Abkürzung	wird angezeigt als
PEEK	P SHIFT E	P
POKE	P SHIFT O	P
PRINT	?	?
PRINT#	P SHIFT R	P
READ	R SHIFT E	R
RESTORE	RE SHIFT S	RE
RETURN	RE SHIFT T	RE
RIGHT$	R SHIFT I	R
RND	R SHIFT N	R
RUN	R SHIFT U	R
SAVE	S SHIFT A	S
SGN	S SHIFT G	S
SIN	S SHIFT I	S
SPC(	S SHIFT P	S
SQR	S SHIFT Q	S
STEP	ST SHIFT E	ST
STOP	S SHIFT T	S
STR$	ST SHIFT R	ST
SYS	S SHIFT Y	S
TAB	T SHIFT A	T
THEN	T SHIFT H	T
USR	U SHIFT S	U
VAL	V SHIFT A	V
VERIFY	V SHIFT E	V
WAIT	W SHIFT A	W

Abkürzungen der Basic-Schlüsselwörter (Basic 7.0), die im Basic des C 64 nicht enthalten sind.

Befehl	Abkürzung		
APPEND	A	SHIFT	P
AUTO	A	SHIFT	U
BACKUP	BA	SHIFT	C
BANK	B	SHIFT	A
BEGIN	B	SHIFT	E
BEND	BE	SHIFT	N
BLOAD	B	SHIFT	L
BOOT	B	SHIFT	O
BOX		KEINE	
BSAVE	B	SHIFT	S
BUMP	B	SHIFT	U
CATALOG	C	SHIFT	A
CHAR	CH	SHIFT	A
CIRCLE	C	SHIFT	I
COLLECT	COLL	SHIFT	E
COLLISION	CO	SHIFT	L
COLOR	COL	SHIFT	O
CONCAT	C	SHIFT	O
COPY	CO	SHIFT	P
DCLEAR	DCL	SHIFT	E
DCLOSE	D	SHIFT	C
DEC		KEINE	
DELETE	DE	SHIFT	L
DIRECTORY	DI	SHIFT	R
DLOAD	D	SHIFT	L
DO		KEINE	
DOPEN	D	SHIFT	O
DRAW	D	SHIFT	R
DSAVE	D	SHIFT	S
DVERIFY	D	SHIFT	V
ENVELOPE	E	SHIFT	N
FAST	F	SHIFT	A
FETCH	F	SHIFT	E
FILTER	F	SHIFT	I
FRE	F	SHIFT	R
GETKEY	GETK	SHIFT	E

Abkürzungen der Basic-Schlüsselwörter (Basic 7.0), die im Basic des C 64 nicht enthalten sind.

Befehl	Abkürzung		
GO64		KEINE	
GRAPHIC	G	SHIFT	R
GSHAPE	G	SHIFT	S
HEADER	HE	SHIFT	A
HELP	HE	SHIFT	L
HEX$	H	SHIFT	E
IF		KEINE	
INSTR	IN	SHIFT	S
JOY	J	SHIFT	O
KEY	K	SHIFT	E
LOCATE	LO	SHIFT	C
MONITOR	MO	SHIFT	N
MOVSPR	M	SHIFT	O
PAINT	P	SHIFT	A
PEN	P	SHIFT	E
PLAY	P	SHIFT	L
POINTER	PO	SHIFT	I
POT	P	SHIFT	O
PRINTUSING	?US	SHIFT	I
PUDEF	P	SHIFT	U
RCLR	R	SHIFT	C
RDOT	R	SHIFT	D
RECORD	R	SHIFT	E
RENAME	RE	SHIFT	N
RENUMBER	REN	SHIFT	U
RESTORE	RE	SHIFT	S
RESUME	RES	SHIFT	U
RGR		KEINE	
RREG	R	SHIFT	R
RSPCOLOR	RSP	SHIFT	C
RSPPOS	R	SHIFT	S
RSPRITE	RSP	SHIFT	R
RUN	R	SHIFT	U
RWINDOW	R	SHIFT	W
SCALE	SC	SHIFT	A
SCNCLR	S	SHIFT	C
SCRATCH	SC	SHIFT	R

Abkürzungen der Basic-Schlüsselwörter (Basic 7.0), die im Basic des C 64 nicht enthalten sind.

Befehl	Abkürzung		
SLEEP	S	SHIFT	L
SLOW	SL	SHIFT	O
SOUND	S	SHIFT	O
SPRCOLOR	SPR	SHIFT	C
SPRDEF	SPR	SHIFT	D
SPRITE	S	SHIFT	P
SPRSAV	SPR	SHIFT	S
SSHAPE	S	SHIFT	S
STASH	S	SHIFT	T
SWAP	S	SHIFT	W
TEMPO	T	SHIFT	E
TRAP	T	SHIFT	R
TRON	TR	SHIFT	O
TROFF	TRO	SHIFT	F
VOL	V	SHIFT	O
WIDTH	WI	SHIFT	D
WINDOW	W	SHIFT	I
XOR	X	SHIFT	O

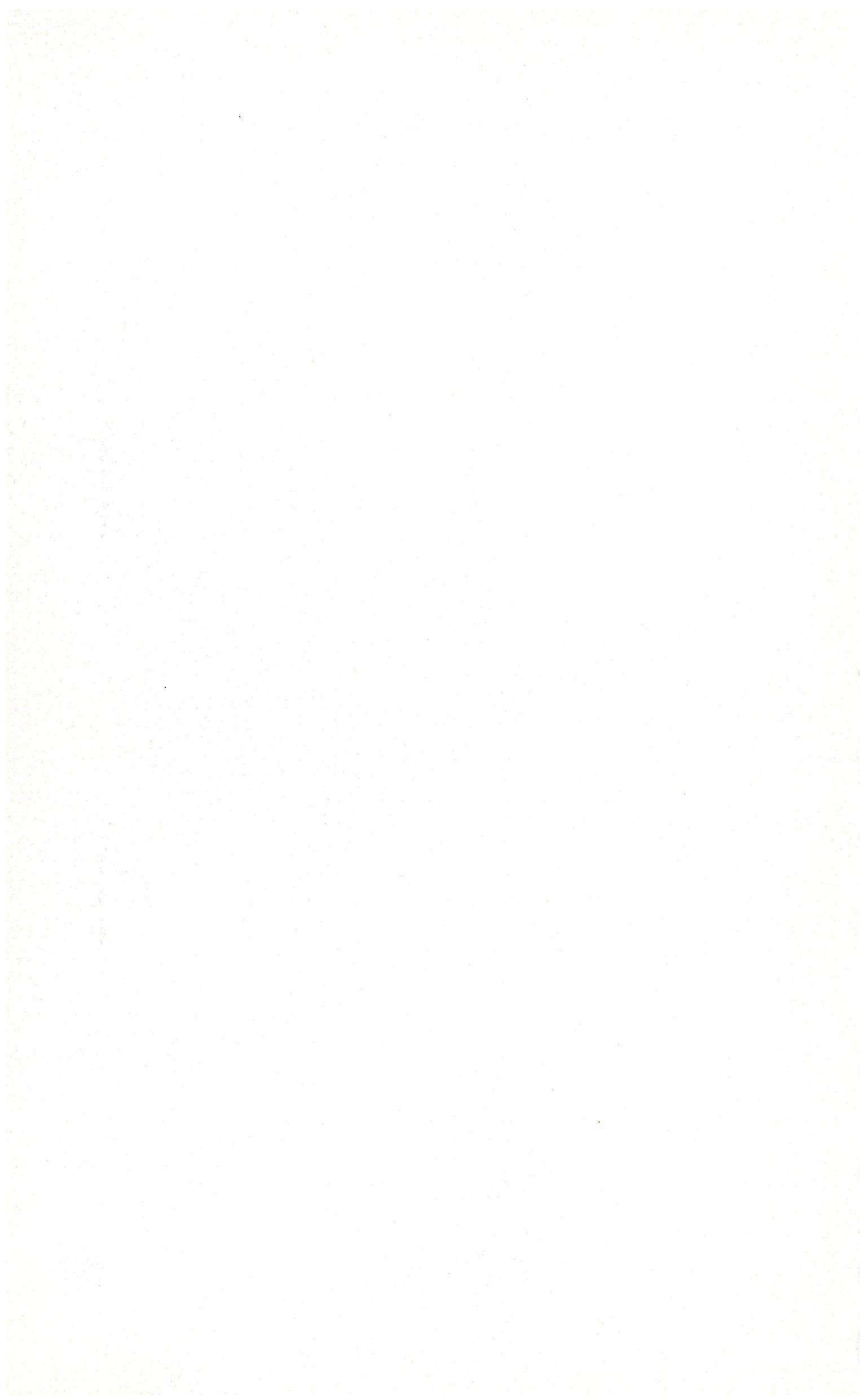
● Anmerkung: Sollten Sie Basic-Schlüsselwörter der Version 2.0 des C 64 im C 128-Modus benutzen, ergeben sich folgende Änderungen:

Keine Abkürzungen möglich für:

CONT
END
SPC
SYS

Neue Abkürzungen ergeben sich für:

PEEK	PE	SHIFT	E
POKE	PO	SHIFT	K
READ	RE	SHIFT	A
STOP	ST	SHIFT	O



Anhang J:

Definition von Tastaturbelegungen

Anhang J: Definition von Tastaturbelegungen

Wie bereits im Kapitel 4.1 bei der Beschreibung der ALT–Taste erwähnt, kann mit dieser Taste eine anwenderspezifische Tastaturbelegung gewählt werden.

Die ALT–Taste arbeitet prinzipiell genauso wie die SHIFT–, C= – oder CTRL–Taste. Wird sie gleichzeitig mit einer weiteren Taste gedrückt, so wird dieser zweiten Taste ein anderer Code zugeordnet. Die den Tasten selbst oder mit der ALT–, SHIFT–, C= –, oder CTRL–Taste modifiziert zugeordneten Codes sind in sogenannten Tastaturtabellen im Speicher abgelegt, wobei standardmäßig die Tabelle für die mit der ALT–Taste modifizierten Tasten dieselbe ist, wie die für die nichtmodifizierten (ohne SHIFT) Tasten. Daher zeigt das Drücken der ALT–Taste zusammen mit einer anderen Taste zunächst keine andere Wirkung, als wenn die Taste alleine gedrückt würde.

Jeder Taste der Tastatur mit Ausnahme der SHIFT–, C= (Commodore)–, ALT–, RESTORE–, ASCII/DIN–, 40/80 DISPLAY– und NO SCROLL–Tasten ist ein Platz in der Tastaturtabelle zugeordnet. Der Code, der an dieser Stelle gespeichert ist, bestimmt, welches Zeichen beim Drücken der zugehörigen Taste angezeigt oder zugeordnet wird.

In der folgenden Tabelle ist die Zuordnung von Tabellenplatz, Taste und Code für die Originaltastaturtabelle der ASCII–Tastatur aufgeführt.

Platz	Taste	Code hex	dez
1	DEL	14	20
2	RETURN	0D	13
3	CRSR rechts	1D	29
4	Funktionstaste f7	88	136
5	Funktionstaste f1	85	133
6	Funktionstaste f3	86	134
7	Funktionstaste f5	87	135
8	CRSR unten	11	17
9	3	33	51
10	w	57	87
11	a	41	65

Tastaturtabelle (Fortsetzung)

Platz	Taste	Code hex	dez
12	4	34	52
13	z	5A	90
14	s	53	83
15	e	45	69
16	reserviert (muß 1 enthalten)	01	1
17	5	35	53
18	r	52	82
19	d	44	68
20	6	36	54
21	c	43	67
22	f	46	70
23	t	54	84
24	x	58	88
25	7	37	55
26	y	59	89
27	g	47	71
28	8	38	56
29	b	42	66
30	h	48	72
31	u	55	85
32	v	56	86
33	9	39	57
34	i	49	73
35	j	4A	74
36	0	30	48
37	m	4D	77
38	k	4B	75
39	o	4F	79
40	n	4E	78
41	+	2B	43
42	p	50	80
43	l	4C	76
44	–	2D	45
45	.	2E	46
46	:	3A	58
47	@	40	64
48	,	2C	44
49	\	5C	92
50	*	2A	42

Tastaturtabelle (Fortsetzung)

Platz	Taste	Code hex	dez
51	;	3B	59
52	HOME	13	19
53	reserviert (muß 1 enthalten)	01	1
54	=	3D	61
55	↑	5E	94
56	/	2F	47
57	1	31	49
58	–	5F	95
59	reserviert (muß 4 enthalten)	04	4
60	2	32	50
61	Leertaste	20	32
62	reserviert (muß 2 enthalten)	02	2
63	q	51	81
64	STOP	03	3
65	HELP	84	132
66	8 (Zehnertastatur)	38	56
67	5 (Zehnertastatur)	35	53
68	TAB	09	9
69	2 (Zehnertastatur)	32	50
70	4 (Zehnertastatur)	34	52
71	7 (Zehnertastatur)	37	55
72	1 (Zehnertastatur)	31	49
73	ESC	1B	27
74	+ (Zehnertastatur)	2B	43
75	– (Zehnertastatur)	2D	45
76	LINE FEED	0A	10
77	ENTER (Zehnertastatur)	0D	13
78	6 (Zehnertastatur)	36	54
79	9 (Zehnertastatur)	39	57
80	3 (Zehnertastatur)	33	51
81	reserviert (muß 8 enthalten)	08	8
82	0 (Zehnertastatur)	30	48
83	. (Zehnertastatur)	2E	46
84	CRSR oben (Cursor–Einzeltaste)	91	145
85	CRSR unten (Cursor–Einzeltaste)	11	17
86	CRSR links (Cursor–Einzeltaste)	9D	157
87	CRSR rechts (Cursor–Einzeltaste)	1D	29
88	reserviert (muß 255 enthalten)	FF	255
89	reserviert (muß 255 enthalten)	FF	255

Wenn Sie nur wenige Tastenbelegungen ändern wollen, ist es am einfachsten, die Originaltabelle, die im ROM liegt, in den RAM–Bereich unterhalb der Adresse $4000 (16384 dezimal) zu kopieren. Dies können Sie z.B. für die ASCII–Originaltabelle mit Hilfe des T–Befehls des Monitors (s.a. Anhang C) folgendermaßen erreichen:

T FFA80 FFAD8 03FA7

Danach ändern Sie entweder mit Hilfe von POKE–Befehlen oder des Monitors die Tabelle nach Ihren eigenen Wünschen ab. Die Originaltabellen für die beiden verfügbaren Tastaturen (ASCII und DIN) sowie die Umschalttasten C= , SHIFT und CTRL liegen im ROM bei folgenden Adressen:

Tabelle		Adresse hex	Adresse dez
Normaltabelle	ASCII	FFA80	Bank 15, 64128
Normaltabelle	DIN	FFD29	Bank 15, 64809
SHIFT–Tabelle	ASCII	FFAD9	Bank 15, 64217
SHIFT–Tabelle	DIN	FFD82	Bank 15, 64898
C= –Tabelle	ASCII	FFB32	Bank 15, 64306
C= –Tabelle	DIN	FFDDB	Bank 15, 64987
CTRL–Tabelle		FFB8B	Bank 15, 64395

Haben Sie die Tabelle nach Ihren Wünschen geändert, so müssen Sie noch den Adreßzeiger der Tastatur–Betriebssystemroutine für die ALT–Taste so ändern, daß er auf den Anfang Ihrer Tabelle zeigt. Der Zeiger steht im RAM bei der Adresse $346,$347 (838,839 dezimal). Haben sie Ihre Tabelle z.B. wie auf der letzten Seite angegeben, ab der Speicheradresse $3FA7 (16295 dez) abgelegt, so können Sie den Adreßzeiger für die ALT–Taste folgendermaßen einstellen:

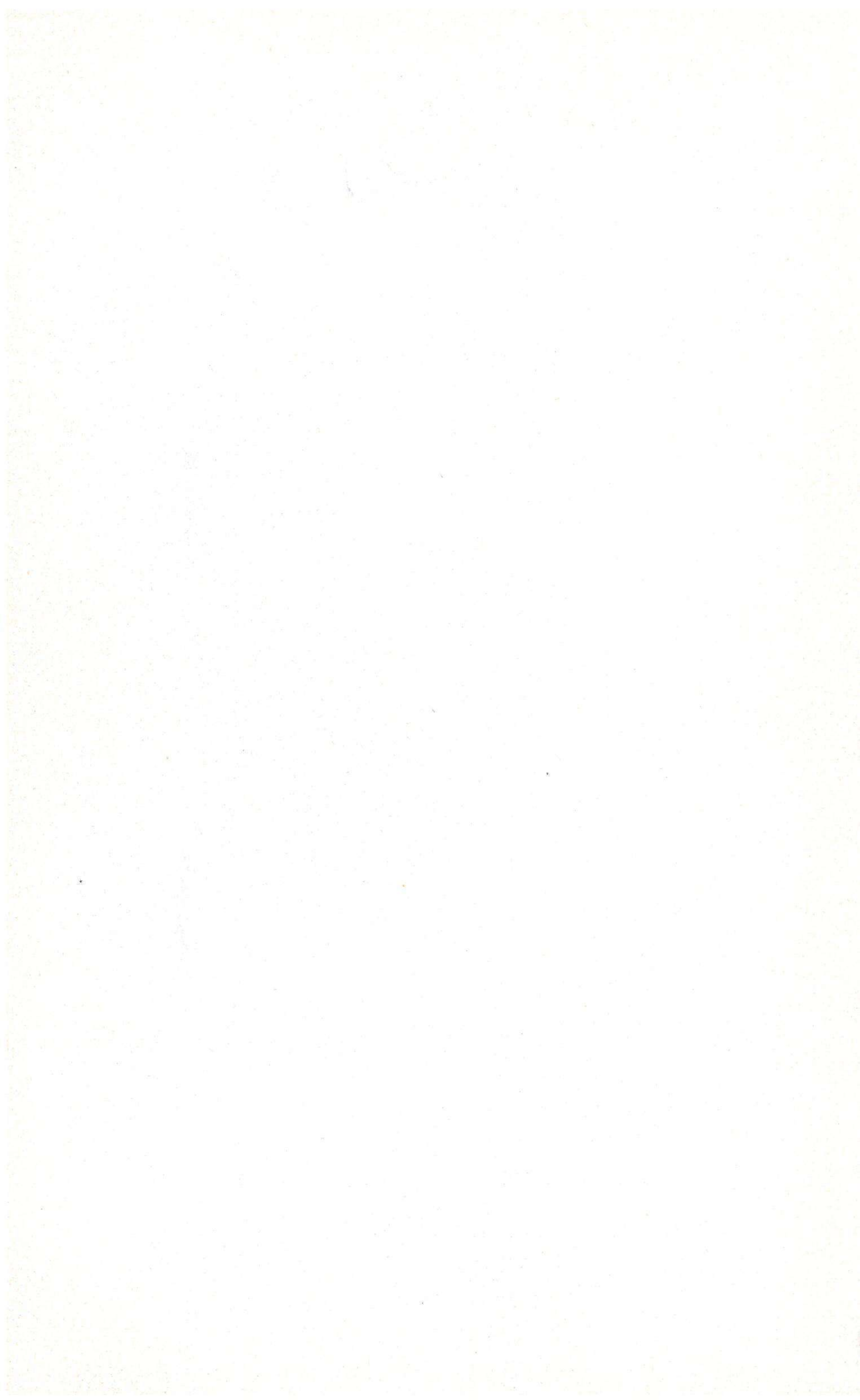
```
BANK 0
POKE 838,16295-INT(16295/256)*256      (niederwert. Teil)
POKE 839,INT(16295/256)                (höherwert. Teil)
```

In die Speicherzelle mit der Adresse 838 wird der niederwertige und in die mit der Adresse 839 der höherwertige Teil der Gesamtadresse eingetragen.

Auf dieselbe Art und Weise wie die ALT–Tastaturbelegung geändert wird, können Sie auch die SHIFT–, C= –, und CTRL–Tastaturbelegung ändern.

Nachfolgend finden Sie die Adreßzeiger für alle Tastaturtabellen zusammengestellt:

Tabelle	Adreßzeiger hex	Adreßzeiger dez
Normaltabelle	33E,33F	830,831
SHIFT–Tabelle	340,341	832,833
C= –Tabelle	342,343	834,835
CTRL–Tabelle	344,345	836,837
ALT–Tabelle	346,347	838,839



Anhang K:

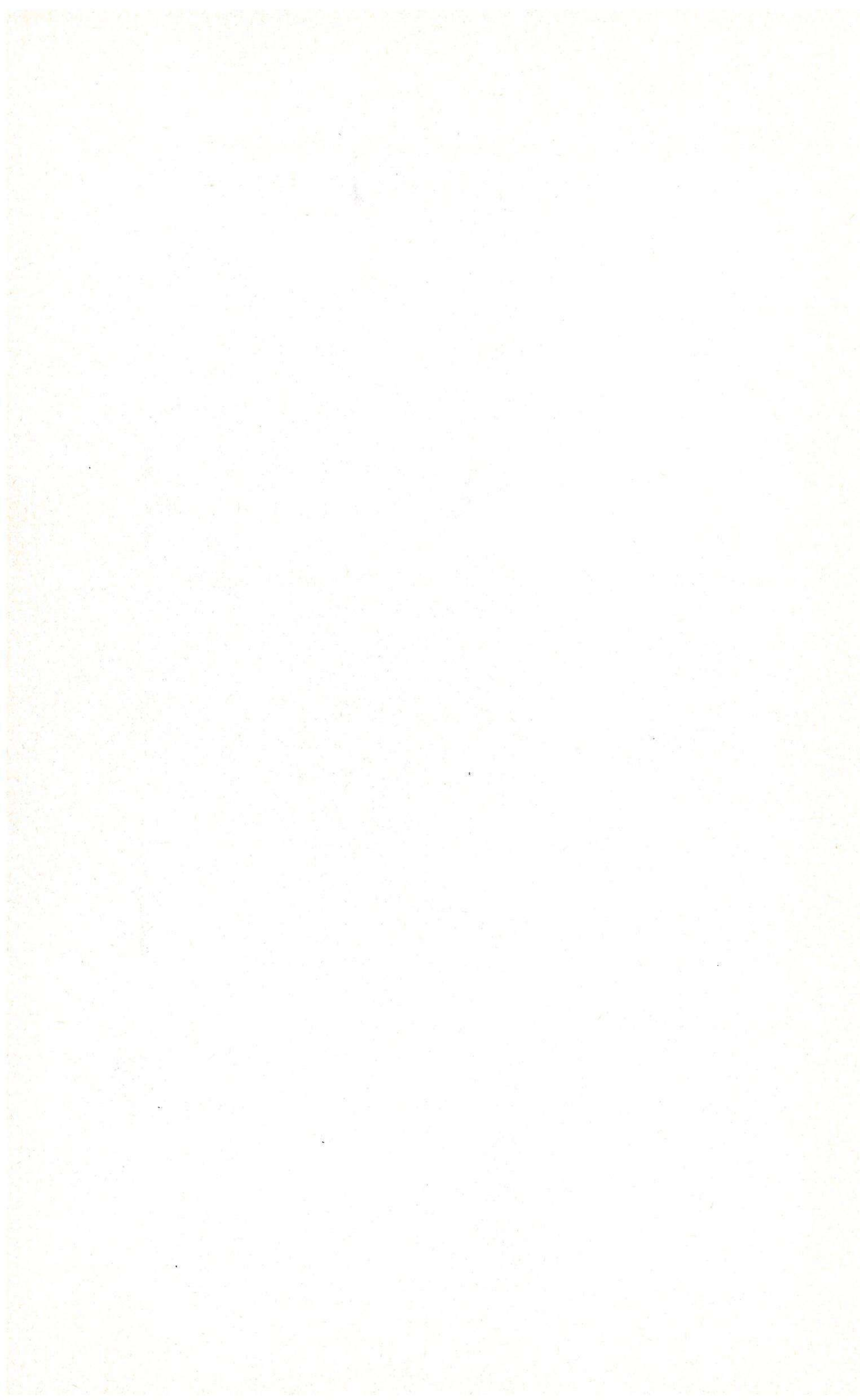
Umschreibung transzendenter Funktionen



Anhang K: Umschreibung transzendenter Funktionen

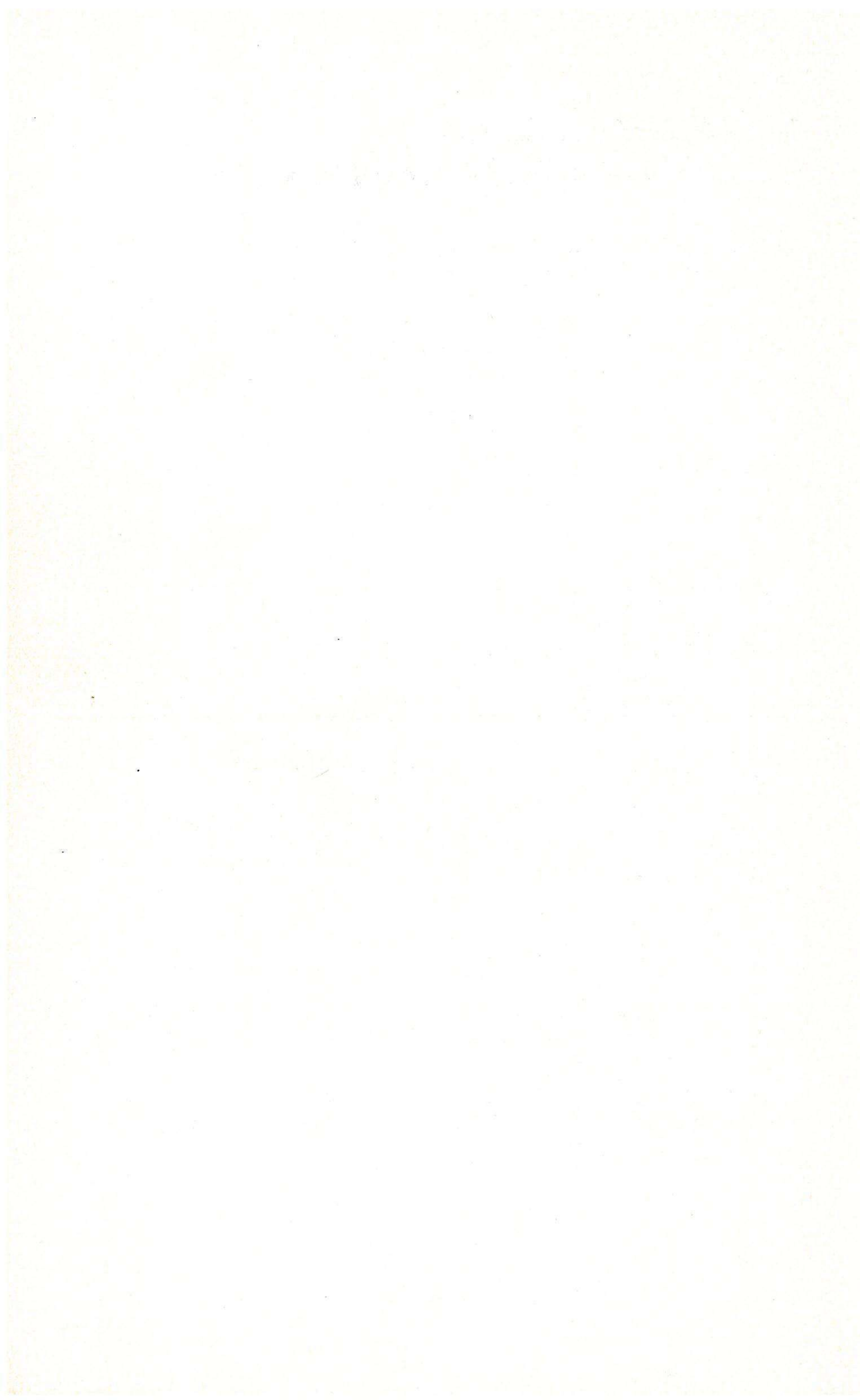
Einige der trigonometrischen, zyklometrischen und alle Hyperbelfunktionen sind nicht Bestandteil des CBM–BASIC und müssen daher wie folgt berechnet werden:

Funktion	Umschreibung in CBM–BASIC
Logarithmus zur Basis B	LOG(X)/LOG(B)
Sekans	1/COS(X)
Cosekans	1/SIN(X)
Cotangens	1/TAN(X)
Arcussinus	ATN(X/SQR(1–X↑2))
Arcuscosinus	1.570796–ATN(X/SQR(1–X↑2))
Arcussecans	ATN(SQR(X↑2–1)) + (X<0)*3.141593
Arcuscosecans	ATN(1/SQR(X↑2–1)) + (X<0)*3.141593
Arcuscotangens	1.57096–ATN(X)
Sinus Hyperbolicus	(EXP(X)–EXP(–X))/2
Cosinus Hyperbolicus	(EXP(X) + EXP(–X))/2
Tangens Hyperbolicus	(EXP(X)–EXP(–X)) /(EXP(X) + EXP(–X))
Cotangens Hyperbolicus	(EXP(X) + EXP(–X)) /(EXP(X)–EXP(–X))
Sekans Hyperbolicus	2/(EXP(X) + EXP(–X))
Cosekans Hyperbolicus	2/(EXP(X)–EXP(–X))
Area Sinus Hyperbolicus	LOG(X + SQR(X↑2 + 1))
Area Cos. Hyperbolicus	LOG(X + SQR(X↑2–1))
Area Tan. Hyperbolicus	LOG((1 + X)/(1–X))/2
Area Cot. Hyperbolicus	LOG((X + 1)/(X–1))/2
Area Sec. Hyperbolicus	LOG((SQR(–X↑2 + 1) + 1)/X)
Area Cosec. Hyperbolicus	LOG((SGN(X)*SQR(X↑2 + 1)/X)



Anhang L:

Steckerbelegungen



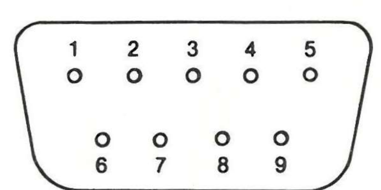
STECKERBELEGUNG DER EIN-/AUSGABE ANSCHLÜSSE

Dieser Anhang soll Ihnen zeigen, wie welches Gerät wo an den C 128 angeschlossen werden kann.

1) Steuereingänge für Spiele
2) Modul-Steckplatz
3) Audio/Video
4) Serielle E/A (Disk/Drucker)
5) Kassette
6) User Port

Control Port 1

Pin	Signal	Bem.
1	JOYA0	
2	JOYA1	
3	JOYA2	
4	JOYA3	
5	POT AY**	
6	BUTTON A/LP*	
7	+5V	MAX. 100mA
8	GND	
9	POT AX**	



Control Port 2

Pin	Signal	Bem.
1	JOYB0	
2	JOYB1	
3	JOYB2	
4	JOYB3	
5	POT BY**	
6	BUTTON B	
7	+5V	MAX. 100mA
8	GND	
9	POT BX**	

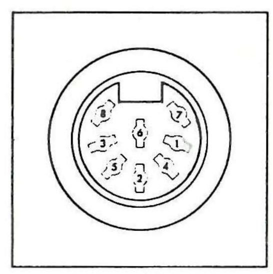
*) Button = Feuerknopf am Joystick
LP = Light pen

**) POT = Paddle Potentiometer

Belegung der Videobuchsen des Commodore 128 Personal Computer

Belegung der 40-Zeichen-Buchse

Pin	Signal
1	LUMINANCE/SYNC
2	MASSE
3	AUDIO OUT
4	VIDEO OUT
5	AUDIO IN
6	CHROMINANCE
7	NICHT ANGESCHLOSSEN
8	5 Volt



Außenaufsicht

Belegung der 80-Zeichen-Videobuchse

PIN	BEZEICHNUNG	BEMERKUNG
1	Masse	
2	Masse	
3	Rot	TTL-Signal, 0-5 Volt Rot-Teil des RGBI-Signals
4	Grün	TTL-Signal, 0-5 Volt Grün-Teil des RGBI-Signals
5	Blau	TTL-Signal, 0-5 Volt Blau-Teil des RGBI-Signals
6	Intensität	TTL-Signal, 0-5 Volt Intensitäts-Teil des RGBI-Signals
7	Monochrome	1 Volt Uss, 15625 Hz Zeilenfrequenz 50 Hz vertikal (BAS)
8	Hsync	TTL-Signal, 0-5 Volt Horizontales Synchronsignal
9	Vsync	TTL-Signal, 0-5 Volt Vertikales Synchronsignal - 50 Hz

5 4 3 2 1
9 8 7 6

Außenaufsicht

Anschluß eines monochromen Monitors

80-Zeichen: 9-polige Sub-D-Buchse
Pin 7 = Signal, Pin 1 = Masse
Analogsignal 1 V Uss, 15625 Hz Zeilenfrequenz

40-Zeichen: (C 64- bzw. C 128-Modus)
8-polige DIN-Video-Buchse Pin 1 =
Luminanz (FBAS) Pin 2 = Masse

Audioanschluß: DIN-Video-Buchse Pin 3

Modul-Steckplatz

Pin	Signal
22	GND
21	CD0
20	CD1
19	CD2
18	CD3
17	CD4
16	CD5
15	CD6
14	CD7
13	$\overline{\text{DMA}}$
12	BA
11	$\overline{\text{ROML}}$
10	$\overline{\text{I/O 2}}$
9	$\overline{\text{EXROM}}$
8	$\overline{\text{GAME}}$
7	$\overline{\text{I/O 1}}$
6	Dot Clock
5	CR/$\overline{\text{W}}$
4	$\overline{\text{IRQ}}$
3	+5V
2	+5V
1	GND

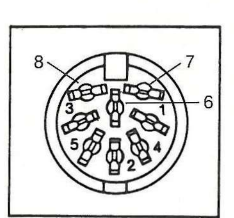
Pin	Signal
Z	GND
Y	CA0
X	CA1
W	CA2
V	CA3
U	CA4
T	CA5
S	CA6
R	CA7
P	CA8
N	CA9
M	CA10
L	CA11
K	CA12
J	CA13
H	CA14
F	CA15
E	$\Phi 2$
D	$\overline{\text{NMI}}$
C	$\overline{\text{RESET}}$
B	$\overline{\text{ROMH}}$
A	GND

22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Z Y X W V U T S R P N M L K J H F E D C B A

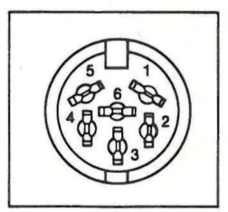
Audio/Video

Pin	Signal	Pin	Signal
1	LUMINANCE	6	CROMINANZ
2	GND	7	O. BELEGUNG
3	AUDIO OUT	8	O. BELEGUNG
4	VIDEO OUT		
5	AUDIO IN		



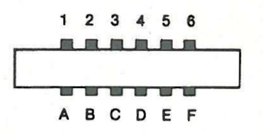
Serielle E/A

Pin	Signal
1	SERIAL $\overline{\text{SRQIN}}$
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	$\overline{\text{RESET}}$



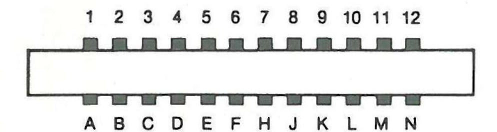
Cassette

Pin	Signal
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE



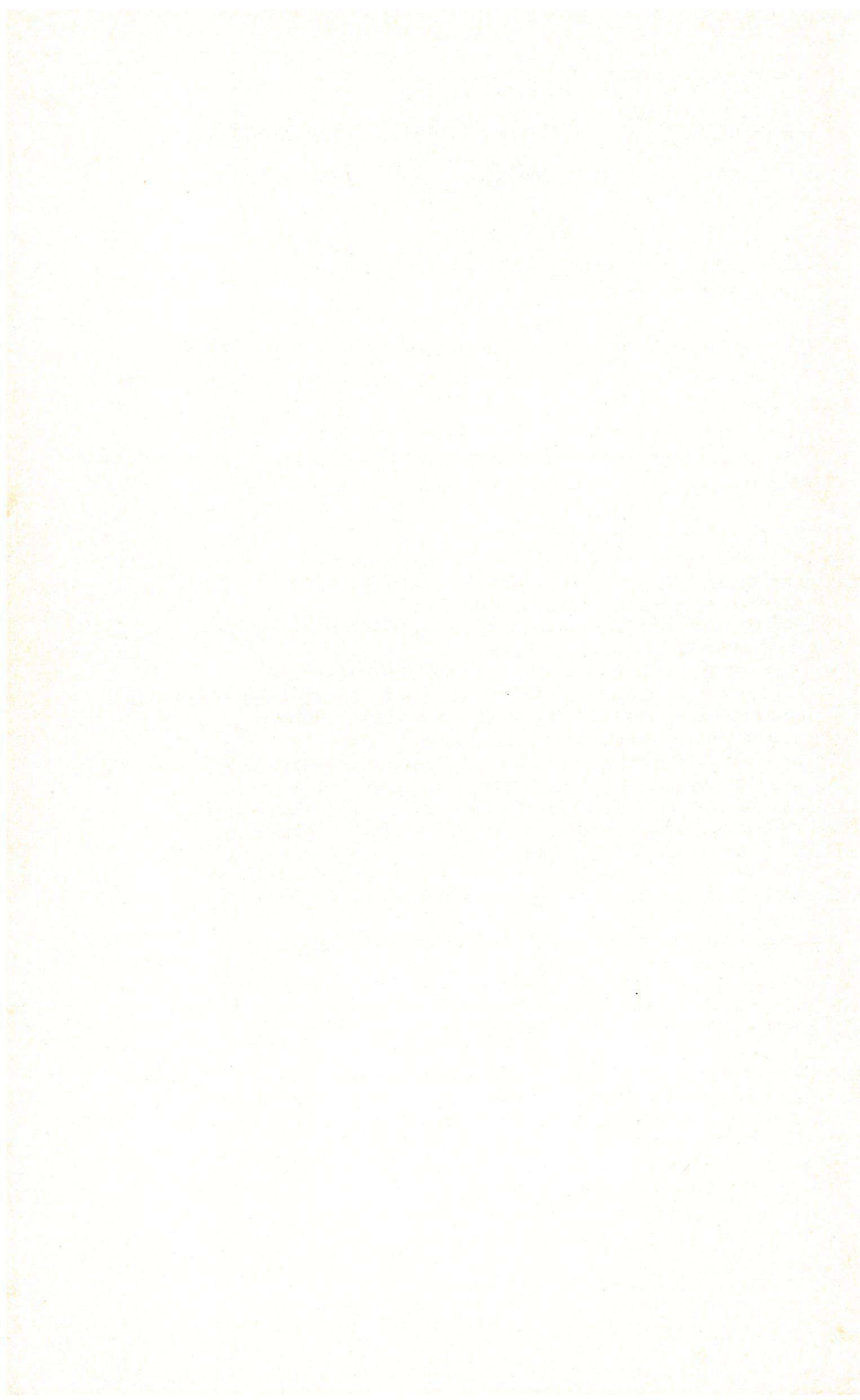
User Port

Pin	Signal	Bemerkung
1	GND	
2	+5V	MAX. 100 mA
3	$\overline{\text{RESET}}$	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	$\overline{\text{PC2}}$	
9	SER. ATN IN	
10	9 VAC	MAX. 100 mA
11	9 VAC	MAX. 100 mA
12	GND	
A	GND	
B	$\overline{\text{FLAG2}}$	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	



Anhang M:

Übertragung von BASIC4-Programmen nach BASIC7



Anhang M: Umwandlung von BASIC4 in BASIC7–Programme

Programm zur Umsetzung von CBM 8000er–Programmen (BASIC 4.0) auf das BASIC 7.0 des C 128.

Es werden aber nur Befehlstokens umgesetzt, nicht jedoch PEEK–, POKE–, SYS– und USR–Adressen; ebenfalls nicht die komplexeren Cursor–Steuerzeichen des BASIC 4.0.

Das Umsetzprogramm selbst ist in BASIC 2 geschrieben, es läuft damit auf jedem beliebigen Commodore–Rechner.

```
10 REM SAVE"0:4NACH7",8
20 REM TOKEN-UMSETZUNG BASIC 4.0 NACH BASIC 7.0
50 DIMA$,ZZ$,N$,FG:N$=CHR$(0):G$=CHR$(34)
100 PRINT"DISKETTE IN DRIVE 0 LEGEN.
105 PRINT"UMSETZUNG ERFOLGT VON DISKETTE ZU DISKETTE
110 INPUT"8XXX-NAME = ";N8$
120 INPUT" 128-NAME = ";N2$:IFN8$=N2$THEN120
140 OPEN2,8,4,N2$:GET#2,A$,A$:S=ST:CLOSE2:REM TESTLESEN
150 IFS=OTHENPRINT"ZIELFILE EXISTIERT!":PRINT:GOTO120
160 OPEN1,8,3,N8$:OPEN2,8,4,N2$+",P,W"
170 GET#1,A$,B$:A8=ASC(A$+N$)+256*ASC(B$+N$):REM LADE-ADR
180 A2=28*256+1:REM FUER 128: LADE-ADR = $1C01
181 H2=INT(A2/256):PRINT#2,CHR$(A2-H2*256);CHR$(H2);
190 DIMC$(256),ZN(256):REM TOKEN-UMSETZ-TABELLEN
191 FORI=204TO256:READA$:IFA$="E"THENI=999:NEXT:GOTO195
192 ZN(I)=-(LEN(A$)>3):C$(I)#CHR$(VAL(LEFT$(A$,3)))
193 IFZN(I)THENC$(I)=C$(I)+CHR$(VAL(RIGHT$(A$,3)))
194 NEXT
200 GET#1,A$,B$:B8=ASC(A$+N$)+256*ASC(B$+N$):
    REM CHAINING-POINTER
201 N8=B8-A8:IFB8=OTHENPRINT#2,N$;N$;:CLOSE1:CLOSE2:
    END:REM PROG-ENDE
210 Z2=0:FG=-1:GET#1,A$:ZZ$=LEFT$(A$+N$,1):REM NEUE
    ZEILE IN ZZ$
211 GET#1,B$:ZZ$=ZZ$+LEFT$(B$+N$,1):REM ZEILENNR. FERTIG
215 PRINT"ZEILE";ASC(LEFT$(ZZ$,1))+256*ASC(RIGHT$(ZZ$,1))
220 FORI=4TON8-2:GET#1,A$:IFA$=G$THENFG=NOTFG:REM FOR
    FUER ZEILE
230 IFFGTHENA=ASC(A$):IFA>203ANDA<255THENA$=C$(A):
    Z2=Z2+ZN(A):REM UMCOD
```

```
240 ZZ$=ZZ$+A$:NEXT:GET#1,A$:ZZ$=ZZ$+N$:REM NULL AM
    ZEILENENDE
250 A8=B8:A2=A2+N8+Z2:H2=INT(A2/256):REM CHAINING
    MITZAEHLEN
260 PRINT#2,CHR$(A2-256*H2);CHR$(H2);ZZ$;:GOTO200:REM
    NEUE ZEILE
9000 REM CONCAT,DOPEN,DCLOSE,RECORD,HEADER,COLLECT
9010 DATA254019,254013,254015,254018,241,243
9020 REM BACKUP,COPY,APPEND,DSAVE;DLOAD,CATALOG
9030 DATA246,244,254014,239,240,254012
9040 REM RENAME,SCRATCH,DIRECTORY
9050 DATA245,242,238,E
```

BESCHEINIGUNG DES HERSTELLERS

Hiermit wird bestätigt, daß der Personal-Computer

COMMODORE C 128

in Übereinstimmung mit den Bestimmungen der

Amtsblattverfügung Nr. 1046/1984

funk-entstört ist.

Der Deutschen Bundespost wurde das Inverkehrbringen dieses Gerätes angezeigt und die Berechtigung zur Überprüfung der Serie auf Einhaltung der Bestimmungen eingeräumt.

COMMODORE BÜROMASCHINEN GMBH

CERTIFICATE OF THE MANUFACTURER

Herewith we certify that our device Personal-Computer

COMMODORE C 128

corresponds to the regulations

Amtsblattverfügung Nr. 1046/1984

is eliminated of radio interference.

The German Bundespost has been informed that this unit is on the market and has got the right to check on the mass production if the limits are kept.

COMMODORE BUSINESS MACHINES LIMITED

Commodore Büromaschinen GmbH,
Lyoner Str. 38, 6000 Frankfurt 71
Tel. (069) 6638-0, Telefax 6638-159
Telex 4185663 como d

Commodore AG,
Aeschenvorstadt 57, CH-4010 Basel,
Tel. (061) 237800, Twx. 64961

Commodore Büromaschinen GmbH,
Kinskygasse 40-44, A-1232 Wien,
Tel. (0222) 675600, Twx. 111350



Art.-Nr.: 580128